VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Emulátor ZX Spectra s integrovaným debuggerem

ZX Spectrum Emulator with an Integrated Debugger

2012 Bc. Miroslav Šustek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Miroslav Šustek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Emulátor ZX Spectra s integrovaným debuggerem
ZX Spectrum Emulator with an Integrated Debugger

Zásady pro vypracování:

Cílem práce je vytvořit emulátor počítače ZX Spectrum, který bude kromě spouštění programů pro tento počítač umožňovat také jejich pohodlné zkoumání a debuggování. Pomocí zabudovaného debuggeru bude možno běh programů pozastavovat a opětovně pouštět, zkoumat obsah paměti a registrů procesoru, disassemblovat instrukce procesoru, nastavovat body přerušení (breakpoints) apod.

1. Detailně nastudujte architekturu počítače ZX Spectrum (instrukční sadu procesoru, organizaci paměti, I/O zařízení, atd.).
2. Sestavte přehled existujících emulátorů ZX Spectra, včetně přehledu možností, které nabízejí.
3. Navrhněte a implementujte emulátor s integrovaným debuggerem se všemi výše popsanými možnostmi:
a) Program bude plně ovládán pomocí GUI.
b) Emulátor bude věrně implementovat veškeré prvky architektury ZX Spectra a to včetně všech známých "nedokumentovaných" vlastností.
c) Emulátor bude podporovat všechny známé formáty souborů používané jinými emulátory pro uložení obsahu paměti (.SNA, .Z80, .ZXS, .SLT, ...) a obsahu magnetofonových kazet s programy (.TAP, .TZX, ...).
d) Emulátor bude implementován jako multiplatformní aplikace primárně určená pro Linux a další systémy unixového typu.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Zdeněk Sawa, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 3. května 2012

Miroslav Gustek

Děkuji svému vedoucímu diplomové práce Ing. Zdeňku Sawovi, Ph.D., za poskytnutí odborných znalostí a za to, že mi umožnil pod svým vedením tuto práci realizovat.

Abstrakt

Tato diplomová práce si klade za cíl nejprve seznámit čtenáře s osobním domácím počítačem ZX Spectrum a vysvětlit základní principy emulace počítačů. Za uplynulé tři dekády vzniklo pro počítač ZX Spectrum nepřeberné množství programů, které mohou být i v dnešní z určitých ohledů zajímavé. Emulátory počítačů umožňují provozovat tyto programy i bez nutnosti vlastnit původní počítač. Druhým cílem práce je popsat emulátor, který vznikl jako součást této diplomové práce a kromě spouštění aplikací určených pro počítač ZX Spectrum umožňuje také zkoumání jejich kódu.

Klíčová slova: ZX Spectrum, Sinclair, emulátor, 8bitový počítač, debugger, wxWidgets, C++

Abstract

Main goal of this thesis is to acquaint the reader with the ZX Spectrum, a personal home computer and to explain basic principles of computer emulation. A countless volume of programmes for the ZX Spectrum was made during the past three decades, which, from some points of view, could be interesting even nowadays. Emulators enables one to run these programmes even without need to own an authentic computer. The second goal of this thesis is to describe the emulator which was developed as a part of the thesis and besides the ability to run applications dedicated for the ZX Spectrum, it makes it possible to explore their code.

Keywords: ZX Spectrum, Sinclair, emulator, 8-bit computer, debugger, wxWidgets, C++

Seznam použitých zkratek a symbolů

CLK	–	Clock
CPU	–	Central Processing Unit
GNU	–	GNU's Not Unix
GPL	–	General Public License
GPU	–	Graphics Processing Unit
GUI	–	Graphical User Interface
HLE	–	High-Level Emulation
HTML	–	Hyper Text Markup Language
INT	–	Interrupt
JSI	–	Jazyk Symbolických Instrukcí
LLE	–	Low-Level Emulation
NMI	–	Non-Maskable Interrupt
OS	–	Operační Systém
PAL	–	Phase Alternating Line
PC	–	Program Counter
PWM	–	Pulse-width Modulation
RAM	–	Random Access Memory
RGB	–	Red-Green-Blue
ROM	–	Read-Only Memory
SP	–	Stack Pointer
ULA	–	Uncommitted Logic Array

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Letošní rok je to právě třicet let od začátku prodeje 8bitového domácího počítače Sinclair ZX Spectrum 48K. Pro mnoho lidí představoval první kontakt se světem digitálních počítačů. Dnes tento počítač, spíše než u někoho doma, najdeme v muzeu výpočetní techniky. Přesto to nutně neznamená, že se musíme rozloučit i se všemi aplikacemi, které pro něj byly za celou dobu své existence vytvořeny. Díky emulaci můžeme tato zachovaná softwarová díla používat i nadále.

Cílem této diplomové práce je vytvořit takový emulátor, který umožní nejen spouštět na dnešních osobních počítačích původní programy určené pro ZX Spectrum, ale také poskytne nástroje pro zkoumání těchto programů z pohledu práce se zdrojovým kódem v jazyce symbolických instrukcí.

Následující kapitola (2) má za cíl čtenáře podrobně seznámit se samotným počítačem Sinclair ZX Spectrum. Tento počítač je svým návrhem poměrně jednoduchý, přesto nejsou informace v tomto textu vyčerpávající, ale uvádí danou problematiku v dostatečném rozsahu pro pochopení dalších kapitol.

Za popisem počítače následuje v kapitole 3 souhrn základních pojmů a faktů okolo emulace počítače a debugování. Následně budou v kapitole 4 popsány již existující emulátory počítače ZX Spectrum a jejich základní rysy. Tento průzkum sloužil jako inspirace pro úvahy, kde lze v této oblasti přinést něco nového nebo co by se dalo vylepšit. V kapitole 5 jsou pak specifikovány požadavky na výsledný systém.

Návrh aplikace je rozdělen celkem do tří částí. První část tvoří návrh samotného emulátoru v kapitole 6. Zde je nejprve počítač „rozebrán“ na jednotlivé součásti (v objektové dekompozici) a poté jsou části složeny zpět jako softwarový systém. Druhou část návrhu tvoří návrh debuggeru. V kapitole 7 je tedy popsáno jak bude debugger plnit požadované funkce a jakým způsobem bude připojen k emulátoru. Tato kapitola se váže na třetí část návrhu aplikace, kterou je grafické rozhraní. V kapitole 9.1 tak bude zmíněno, jak probíhá vzájemná komunikace mezi grafickým rozhraním a emulátorem s debuggerem.

Implementace podstatných funkcí systému je rozebrána v kapitole 8. Na začátku této kapitoly (v podkapitole 8.1) jsou ale nejprve zmíněny veškeré části systému, jež pochází od třetích stan. Tím jsou myšleny knihovny nebo části zdrojových kódů, které nevznikly v rámci této práce, ale jsou v ní využity.

V kapitole 10 jsou stručně zmíněny techniky, jakými byla aplikace testována, a také jsou uvedeny výsledky některých testů. Především je zde kladen důraz na nedostatky, které se v rámci této práce nepodařilo odstranit.

2 Počítač Sinclair ZX Spectrum

Sinclair ZX Spectrum je 8bitový počítač, který byl vyráběn v průběhu let 1982-1992 firmou Sinclair Research na území Velké Británie. Počítač vešel na trh v éře velkého rozvoje domácích počítačů. Konkurencí pro něj v té době byly především počítače amerických výrobců Atari a Commodore. Sice nebyl v této velké konkurenci nejprodávanějším, ale i tak, nejspíše díky příznivé své ceně, která při začátku prodeje činila 125 liber, dosáhl na tehdejší dobu úctyhodných pěti miliónu prodaných kusů. Pro srovnání, vůbec nejprodávanějším 8bitovým počítačem byl (dle [7]) Commodore 64, jehož celkový prodej se odhaduje na 17 miliónů kusů. O popularitě ZX Spectrum vypovídá také to, že pro něj vzniklo přes dvacet tisíc aplikací, z nichž většinu tvoří hry (podle [12]).

Existuje několik variant počítače ZX Spectrum, které se krom vzhledu liší hlavně množstvím dostupné operační paměti RAM a vestavěným programovým vybavením. Pakliže nebude výslovně uvedeno jinak, budu se v této práci zabývat pouze modelem Sinclair ZX Spectrum 48K (na obrázku 1).

Firma Sinclair dále vyrobila modely ZX Spectrum+ a ZX Spectrum 128. V roce 1986 byla značka „Sinclair“ zakoupena britskou firmou Amstrad, která pokračovala modely ZX Spectrum +2, +2A, +2B a +3.

Kromě oficiálních modelů však po celém světě vznikaly také tzv. *kolny*, vyráběné různými firmami nebo i jednotlivci, které více či méně kopírovaly architekturu modelů ZX Spectrum. Za zmínku stojí například počítač Pentagon, velice podobný ZX Spectrum 128, který byl vyráběn v SSSR. V České republice byl asi nejvíce známý počítač Didaktik M, klon ZX Spectrum 48K, vyráběný od roku 1990 na území dnešní Slovenské republiky.

Obrázek 1: Počítač Sinclair ZX Spectrum 48K

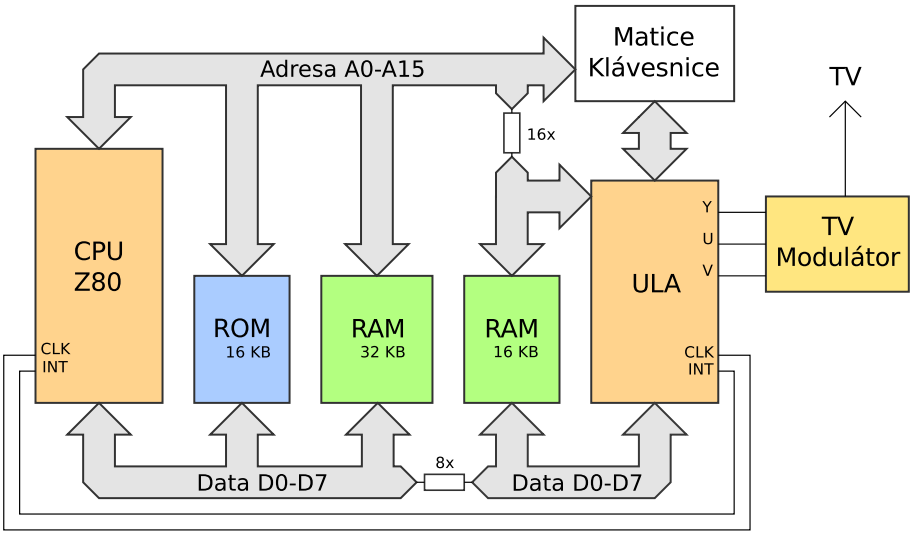
ZX Spectrum 48K, jak již název napovídá, je model s 48 kilobajty paměti RAM. Počítač má ve svém plastovém těle zabudovanou gumovou klávesnici. Jako zobrazovací jednotka slouží klasický televizor, který je možné připojit pomocí anténního konektoru na zadním panelu přístroje. Obrazový výstup je barevný, ovšem s jistými omezeními (podrobněji v podkapitole 2.5). Pro nahrávání a ukládání programů se používají magnetofonové pásky. V následujících podkapitolách budou podrobněji rozebrány hardware a architektura počítače.

2.1 Architektura

Architektura počítače částečně inspirována Von Neumannovým schématem, neboť je zde společná systémová sběrnice pro data i programový kód (dále jen *datová sběrnice*). Datová sběrnice má šířku 8 bitů, zatímco adresní sběrnice je 16bitová.

U obou sběrnic je užito důmyslné řešení, kdy je pomocí rezistorů sběrnice rozdělena na dvě části. Na blokovém schématu počítače (obrázek 2) lze spatřit, že adresní spolu s datovou sběrnicí rozdělují komponenty počítače do dvou skupin. Vlevo jsou to CPU, paměť ROM a horních 32 KB paměti RAM, vpravo čip ULA a spodních 16 KB paměti RAM. Díky tomuto zapojení může procesor pracovat s oblastmi paměti na své části sběrnice, zatímco jednotka ULA čte obrazová data z jiné oblasti paměti.

Pokud dojde k tomu, že se procesor snaží přistoupit do spodních 16 KB paměti RAM ve chvíli, kdy ULA potřebuje číst obrazová data, má jednotka ULA přednost a pozastaví činnost procesoru dočasným odstavením jeho hodinového signálu `CLK`.



Obrázek 2: Blokové schéma počítače

2.2 Procesor – CPU

Jako procesor je v počítači použit Zilog Z80 taktovaný na kmitočtu $3,5\ MHz$. Tento procesor svým návrhem vychází z procesoru 8080 firmy Intel a je s ním zpětně kompatibilní.

V následujících podkapitolách jsou shrnuty základní rysy procesoru. Podrobné detaily lze nalézt v oficiálním manuálu k procesoru[19]. Odborná veřejnost zkoumáním procesoru objevila množství operačních kódů instrukcí a dalších vlastností, které nejsou publikovány v rámci oficiálního manuálu. Na tento popud vnikl neoficiální manuál[6], který mapuje většinu těchto nalezených specifik.

V následujících podkapitolách jsou shrnuty pouze základní rysy procesoru Zilog Z80, které by měli čtenáři usnadnit pochopení práce procesoru.

2.2.1 Registry

Procesor Z80 stejně jako jeho vzor disponuje sadou 8bitových a 16bitové registrů. Některé dvojice 8bitových registrů lze také použít v páru jako jeden 16bitový registr.

V tabulce 1 je uvedena základní sada registrů, která byla dostupná již u procesoru Intel 8080. Speciálně registr příznaků F se lze dále rozdělit až na jednotlivé bity, které uchovávají informace o provedených instrukcích (zejména aritmetických a logických).

Registry	**Funkce**
A	8bitový střadač (*Accumulator*)
F	registr příznakových bitů (*Flags*)
BC	16bitový datový/adresní registr, čítač či dva 8bitové registry B a C
DE	16bitový datový/adresní registr či dva 8bitové registry D a E
HL	16bitový střadač/adresní registr či dva 8bitové registry H a L
SP	16bitový ukazatel vrcholu zásobníku (*Stack Pointer*)
PC	16bitový adresa prováděné instrukce (*Program Counter*)

Tabulka 1: Základní sada registrů procesoru Zilog Z80

Registry	**Funkce**
IX, IY	dva 16bitové indexové registry
I	8bitový ukazatel stránky přerušení
R	8bitový registr pro obnovení dynamické paměti RAM
AF', BC', DE', HL'	stínové kopie registrů A, F, BC, DE a HL

Tabulka 2: Nové registry přidané oproti procesoru Intel 8080

2.2.2 Instrukční sada

Instrukční sadu procesoru tvoří (dle [19]) celkem 158 různých instrukcí. Tyto instrukce zahrnují operace jako je načítání z a ukládání do paměti, aritmetické a logické operace, bitové operace, vstupně-výstupní operace či instrukce větvení pomocí skoků a volání.

Pro orientaci v následujícím textu si vystačíme se znalostí instrukcí `LD`, `ST`, `IN` a `OUT`. První dvě zmíněné instrukce slouží pro načítání dat z paměti (`LD` – *Load*) a ukládání do paměti (`ST` – *Store*). Druhá dvojice instrukcí slouží pro vstupní požadavek (`IN` – *Input*) a výstupní požadavek (`OUT` – *Output*) zaslaný vstupně-výstupním zařízením.

Každá instrukce je v paměti počítače uložena jako sekvence jednoho až čtyř bajtů, kdy počáteční bajty tvoří operační kód instrukce a za nimi může volitelně následovat několik bajtů, které reprezentují operandy instrukce.

2.2.3 Běh a přerušení

Procesor spouští programový kód po jednotlivých instrukcích. Zpracování každé instrukce se dělí do několika tzv. *strojových cyklů* (*M-cycles*). Při prvním strojovém cyklu (označovaný jako `M0`) při zpracování instrukce se podle adresy v registru `PC` načte operační kód instrukce. Registr `PC` je poté inkrementován, aby ukazoval na adresu následující instrukce. Následně je kód instrukce dekódován a provedena požadovaná operace (například načtení jednoho bajtu z paměti do registru). Jednoduché instrukce jsou vykonány přímo během cyklu `M0`. Složitější operace jsou procesorem rozloženy do více následných strojových cyklů. Každý strojový cyklus navíc může mít různou délku, která se měří v počtu *hodinových cyklů* (také *T-states*) zdroje časování procesoru.

Postupné zpracování instrukcí může být narušeno dvěma externími zdroji přerušení přivedenými na odpovídající pin procesoru `INT` nebo `NMI`. Hlavní rozdíl mezi těmito typy přerušení je ten, že zatímco přerušení `INT` lze softwarově zakázat (*maskovat*), `NMI` je nemaskovatelné.

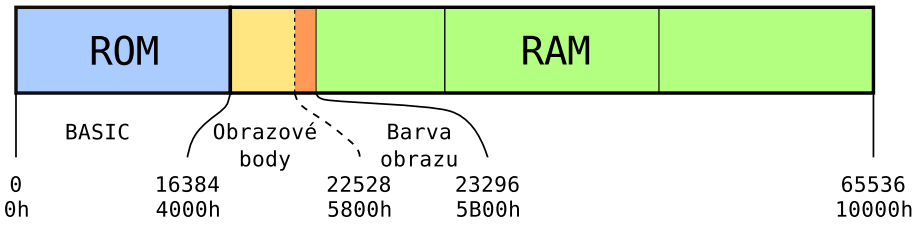
Pokud je na začátku strojového cyklu `M0` detekován a přijat požadavek na přerušení, je procesorem uložen aktuální stav a namísto instrukce se zpracuje toto přerušení. Po návratu z přerušení je obnoven dříve uložený stav a procesor pokračuje v klasickém vykonávání instrukcí.

2.3 Organizace paměti

Procesor svou 16bitovou adresní sběrnicí dokáže adresovat celkem 2^{16} bajtů, tedy 64 kilobajtů. Každý ze čtyř úseků paměti o 16 KB tvoří tzv. *stránku paměti*. Paměť počítače lze dle funkce rozdělit na dvě části – paměť ROM a paměť RAM. Paměť ROM tvoří prvních 16 KB adresního prostoru procesoru. Do této části paměti není možno zapisovat. Nachází se zde vestavěný software s editačním rozhraním a interpretrem programovacího jazyka BASIC.

Zbývajících 48 KB paměti tvoří paměť RAM. Do této oblasti již lze také zapisovat, ovšem ne celou paměť RAM je možno užívat k libovolným účelům. Speciálně adresy 16384-23295 (neboli `4000h-5B00h` v hexadecimální soustavě) jsou určeny pro obrazovou

paměť, odkud obvod ULA průběžně čte obrazová data a převádí je na videosignál (viz podkapitola 2.5).



Obrázek 3: Mapa paměťového prostoru počítače

2.4 Zakázkový čip – ULA

Velmi důležitou komponentou počítače je čip ULA (*Uncommitted Logic Array*). Tento obvod zastává funkci jakéhosi komunikačního centra celého počítače. Stará se o generování obrazu (kap. 2.5), zvukový výstup na vnitřní reproduktor (kap. 2.6.3), dále zprostředkovává komunikaci s magnetofonem (kap. 2.6.2) nebo čtení stisků tlačítek klávesnice (kap. 2.6.1).

Jednotka ULA posílá procesoru s frekvencí $50\ Hz$ požadavek na přerušení `INT`. Toto přerušení jednak spouští obslužné rutiny v paměti ROM a také je v programech (pro svou pravidelnost) požíváno k synchronizaci (pomocí instrukce `HALT`).

Mimo signálu přerušení čip ULA generuje dokonce samotný hodinový signál procesoru. V počítači je krystalovým oscilátorem o frekvenci $14\ MHz$ přiváděn hodinový signál do jednotky ULA, kde je vydělen čtyřmi na $3,5\ MHz$ a poté přiveden na vstup hodinového signálu procesoru – `CLK`.

2.5 Generování obrazu

Jak již bylo zmíněno dříve v této kapitole, o generování obrazového a zvukového signálu se stará čip ULA. Tento obvod padesátkrát do vteřiny přečte obsah obrazové části paměti RAM a převede tato data na tři složky barevného video signálu Y'UV. Ten je dále modulátorem zpracován na televizní signál PAL a přiveden na anténní výstup počítače.

Frekvence $50\ Hz$ pro čtení obrazu z paměti vychází právě z televizní normy PAL, která sice přenáší 25 snímků za vteřinu, ovšem jedná se o signál prokládaný. To znamená, že každý z 25 snímků se přenáší jako dva *půlsnímky* – jeden půlsnímek pouze se sudými řádky, druhý pouze s lichými řádky obrazu.

2.5.1 Obrazové body

Obraz se skládá z 256x192 bodů jejichž barevná paleta nabízí 8 barev navíc s možností dvou stupňů jasu. Okolí obrazových bodů ještě vyplňuje okraj (tzv. *border*; na obrázku 4 šedou barvou), který může mít pouze jednu z osmi základních barev (viz levý sloupec v tabulce 6).

Ani obrazové body ovšem nemohou mít libovolné barvy z této 16barevné palety. Obraz je rozdělen na *bloky* o rozměru 8x8 bodů a v každém z nich mohou mít body jen jednu ze dvou nastavených barev – *barvu kresby* (angl. *ink*) nebo *barvu pozadí* (*paper*).

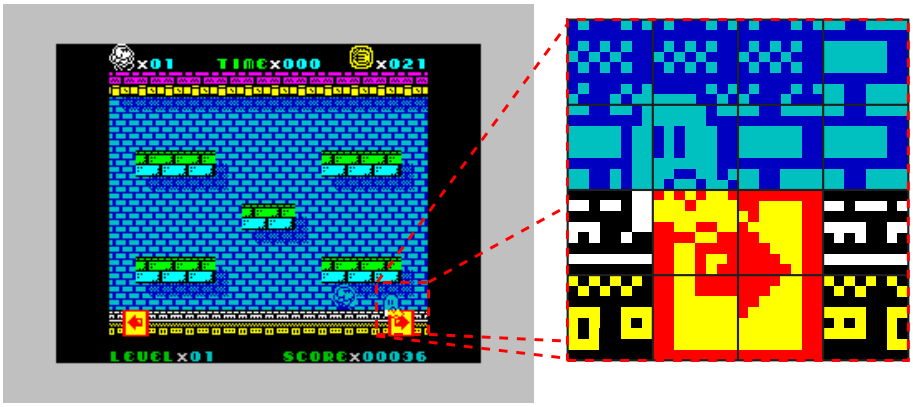
Každý obrazový bod zabírá v paměti jediný bit, kterým je určeno, právě zda má bod barvu kresby resp. barvu pozadí (bit má hodnotu jedna resp. nula).

V jednom bajtu je uloženo osm vodorovně po sobě jdoucích bodů. Pro vysvětlení pořadí v jakém jsou za sebou bajty uloženy v paměti zavedeme několik pojmů. Pojmem *linka* budeme označovat body jdoucí horizontálně vedle sebe (pruh vysoký jeden bod). Jako *řádek* nazveme osm linek pod sebou, neboli vodorovný pruh o výšce osmi bodů. *Sloupec* pak budeme chápat jako svislý pruh široký osm bodů. Řádky a sloupce tedy tvoří na obrazovce mřížku 32x24 osmibodových bloků.

Jednu linku tvoří 32 po sobě jdoucích bajtů. Za první linkou bodů však v paměti nenásleduje druhá obrazová linka, jak by se dalo předpokládat. Obrazovka je totiž vertikálně rozdělena do tří částí po osmi řádcích (64 linkách). V obrazové paměti je nejprve uložena první (horní) třetina, poté druhá (prostřední) a nakonec třetí (dolní). V rámci třetiny jsou dále prokládány linky jednotlivých řádků. Nejprve první linky všech osmi řádků třetiny, poté druhé linky, atd.

V paměti na adrese 16384 (`4000h`) tedy začíná linka 0, poté linky 8, 16, 24, 32, 40, 48 a 54. Za nimi následují linky 1, 9, 17, 25, 33, 41, 49, 55. Teprve po všech linkách z první třetiny obrazovky následují dle obdobného schématu linky druhé a poté třetí třetiny.

Vztah mezi adresou bajtu v paměti a pozicí bodů na obrazovce je zachycena tabulkou 3. Třetiny, řádky, linky i sloupce jsou zde číslovány od nuly.



Obrázek 4: Detail grafiky na obrazovce počítače
(vlevo snímek obrazovky ze hry *Uwol, Quest for Money*,
vpravo 4x4 bloky po 8x8 bodech)

Adresní bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Hodnota	0	1	0	třetina		linka			řádek			sloupec				

Tabulka 3: Adresa linky obrazových bodů v paměti

2.5.2 Barevné atributy

Za oblastí s obrazovými body v paměti následuje na adrese 23296 (`5B00h`) oblast barvy obrazu neboli *barevných atributů*. Zde každému bloku 8x8 obrazových bodů odpovídá jeden bajt s nastavením barev. Význam jednotlivých bitů je naznačen v tabulce 4.

Dolní tři bity (0-2) určují barvu kresby a následující tři bity (3-5) barvu pozadí (označení G, B a R odpovídá postupně zelené, modré a červené složce barevného modelu RGB). Bit 6 nastavený na jedničku zvyšuje jas obou barev. Nastavení posledního bitu (7) na jedničku způsobí, že se při zobrazování bloku na obrazovce mezi sebou střídají barvy kresby a pozadí. Toto blikání (*flash*) má periodu 32 obrazových půlsnímků (každých 16 půlsnímků – 0,32 sekund – dojde k výměně barev).

Barva obrazu, na rozdíl od obrazových bodů, není v paměti ukládaná po linkách ani třetinách, ale jednoduše jako řádky a sloupce bloků 8x8 bodů. Tabulka 5 zachycuje, jak je tvořena adresa bloku.

Barva okraje obrazovky není jednotkou ULA načítána odnikud z paměti, nýbrž je nastavována procesorem do ULA a to zápisem na port č. 254 (`FEh`) jak je možno vidět v tabulce 7.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Označení	FLASH	BRIGHT	G	R	B	G	R	B
Význam	blikání	zvýšený jas	barva pozadí			barva kresby		

Tabulka 4: Význam bitů v bajtu s barevnými atributy obrazového bloku

Adresní bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Hodnota	0	1	0	1	1	0	řádek					sloupec				

Tabulka 5: Adresa barvy bloku v paměti

Barva	**Zvýšený jas**	**Dekadicky**	**Binárně**
černá		0	000
modrá		1	001
červená		2	010
purpurová		3	011
zelená		4	100
azurová		5	101
žlutá		6	110
bílá		7	111

Tabulka 6: Barevná paleta počítače

2.6 Vstupně-výstupní zařízení

Kromě televizní obrazovky má počítač ještě několik dalších vstupně-výstupních zařízení. V následujících podkapitolách budou popsány jednak klávesnice a reproduktor, které jsou zabudovány přímo v těle počítače, a také nejběžnější externí vstupně-výstupní zařízení – magnetofon a joystick.

Se vstupně-výstupními zařízeními procesor komunikuje pomocí instrukce `IN` resp. `OUT`, která vyvolá na datové sběrnici požadavek na čtení resp. zápis na nějaké zařízení. S každým požadavkem je na adresní sběrnici nastavena adresa zařízení (také zvaná *port*), které má na požadavek reagovat.

Speciální pozici vstupně-výstupního zařízení zastává jednotka ULA, která reaguje na všechny `IN` a `OUT` požadavky na sudých adresách (číslo portu má nejnižší bit nulový). Nejčastěji se však pro adresování jednotky ULA používá port s nižším bytem rovným 254 (`FEh`).

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Čtení (`IN`)	–	–	EAR	KB4	KB3	KB2	KB1	KB0
				řádky matice klávesnice				
Zápis (`OUT`)	–	–	–	SPK	MIC	G	R	B
						barva okraje obrazovky		

Tabulka 7: Význam datových bitů při vstupně-výstupních operacích na portu 254

2.6.1 Klávesnice

Klávesnice je hlavním uživatelským nástrojem pro ovládání počítače. Klávesy s písmeny jsou rozmístěny dle schématu QWERTY, které je běžné i dnes. Horní řadu kláves tvoří číslice 1-9 a 0. Dále jsou zde čtyři speciální klávesy – ENTER (nový řádek), CAPS SHIFT (psaní velkých písmen), SYMBOL SHIFT (alternativní funkce kláves) a SPACE (mezera).

Uvnitř počítače je klávesnice tvořena membránou, která spojuje klávesy do matice 5x8 kláves. Každý z pěti řádků matice je jeden z vodičů `KB0-KB4`, které jsou prostřednictvím jednotky ULA propojovány na vodiče datové sběrnice `D0-D4`. Sloupce matice pak tvoří vodiče adresní sběrnice `A8-A15` (viz tabulka 8). Stisknutím tlačítka dojde k propojení příslušného vodiče v řádku a sloupci matice.

Počítač zjišťuje která klávesa je stisknutá postupným skenováním po jednotlivých vodičích `A8-A15`. Nejprve je tedy přiveden signál například na vodič `A8` a poté se z datové sběrnice přečte, zda je některá z kláves CAPS SHIFT, Z, X, C nebo V stisknutá. Je potřeba objasnit, že přivedení signálu v tomto případě znamená přivedení logické nuly na tento vodič. Vodiče obou sběrnic počítače totiž mají v klidovém stavu hodnotu logické jedničky.

Skenování je realizováno instrukcemi `IN`, které mají nižší adresní bajt roven 254 (FEh). Takovýto požadavek je vyřízen jednotkou ULA, která propojí vodiče datové sběrnice `D0-D4` s vodiči řádků matice klávesnice `KB0-KB4`. Horní adresní bajt, který je přiveden na sloupce matice vybírá jeden z vodičů. Například požadavek `IN` na adrese `01FEh` načte

Vodič	A8	A9	A10	A11	A12	A13	A14	A15
KB0 (D0)	CAPS SHIFT	A	Q	1	0	P	ENTER	SPACE
KB1 (D1)	Z	S	W	2	9	O	L	SYMBOL SHIFT
KB2 (D2)	X	D	E	3	8	I	K	M
KB3 (D3)	C	F	R	4	7	U	J	N
KB4 (D4)	V	G	T	5	6	Y	H	B

Tabulka 8: Matice klávesnice počítače

bajt, který bude mít nulové bity na pozicích odpovídajících stlačeným klávesám z prvního sloupce `A8`.

2.6.2 Magnetofon

Magnetofon, který se k počítači připojuje přes konektory *MIC* a *EAR*, složí jako trvalé úložiště pro aplikace a data (dále jen data). Data jsou na magnetické pásce uložena jako obdélníkový zvukový signál. Pro uložení binárních dat je použita pulsně-šířková modulace (*PWM*), kdy je každý bit zakódován jako jedna celá perioda (kladná a záporná půlperioda) obdélníkového signálu o délce přibližně $244\ \mu s$ pro nulový bit, nebo $489\ \mu s$ pro bit jedničkový.

Pro načítání programu z pásky je výstup z magnetofonu je připojen na zvukový vstup počítače EAR[1], odkud je signál přiveden do jednotky ULA, která provádí 1bitový převod analogového signálu na digitální. Tento digitální signál je procesorem z ULA čten operací `IN` na portu 254, kde je dostupný v bitu 5 (viz tabulka 7). Tento signál, přestože je již binární, je stále zakódován pulsně-šířkovou modulací. Převod na data je prováděn až softwarově podprogramem v paměti ROM.

Ukládání dat na pásku probíhá opačným způsobem. Nejprve se softwarově data zakódují do pulsů jedniček a nul, které jsou výstupními operacemi `OUT` na port 254 zapisovány do jednotky ULA v bitu 3 (viz tabulka 7). ULA pak už jen upraví úrovně signálu a přivede jej na výstupní konektor MIC, který je připojen k mikrofonního vstupu magnetofonu.

[1] Ač by se podle názvu dalo předpokládat, že tento konektor je zvukovým výstupem počítače (z anglického *earphone* – sluchátko), tak dle oddílu 5.4.1 v [10] se skutečně jedná o zvukový vstup. Nejspíše je konektor takto označen proto, že se do něj připojuje sluchátkový výstup z magnetofonu.

2.6.3 Vnitřní reproduktor

Výstup na vnitřní reproduktor (*speaker*) je jednobitový a je podobně jako u magnetofonu realizován zápisem na port 254, konkrétně na bit 4 označovaný také jako *SPK* (viz tabulka 7). Ve skutečnosti jsou však výstup na vnitřní reproduktor a na výstupní i vstupní konektor (MIC a EAR) vnitřně přenášeny po jediné lince. Nikdy se však v počítači nepoužívá najednou zvukový vstup i výstup (není možné zároveň číst z kazety a zapisovat na ni nebo přehrávat zvuk). Bity 3 a 4 výstupního portu 254 tedy mají v podstatě stejnou funkci. Rozdíl je v tom, že při použití bitu 4 je produkována vyšší amplituda výstupního signálu než při použití bitu 3.

2.6.4 Joystick

Vedle klávesnice lze jako vstupní zařízení použít také joystick, který se připojuje přes konektor na zadním panelu počítače. Existuje několik typů joysticků, které se liší způsobem, jakým s počítačem komunikují. Nejpoužívanějším typem je pravděpodobně joystick *Sinclair*, který pracuje jako vstupní zařízení odpovídající na portu `F7FEh`.

Joystick Sinclair má celkem 5 tlačítek, kdy každé z nich při stisknutí nastaví jeden bit na datové sběrnici na nulu. Pokud tlačítko stisknuté není, ponechává patřičný bit datové sběrnice v klidovém stavu – hodnotě jedna. V tabulce 9 je znázorněno, které bity jsou ovlivňovány jednotlivými tlačítky joysticku. Je dobré poznamenat, že tlačítka joysticku Sinclair vlastně odpovídají klávesám klávesnice ve sloupci `A11` (viz tabulka 8).

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Tlačítko	–	–	–	střelba	nahoru	dolů	vpravo	vlevo

Tabulka 9: Mapování tlačítek joysticku Sinclair na datové bity

2.7 Software

Jedním z klíčových faktorů velké popularity počítače firmy Sinclair bylo široké spektrum dostupných aplikací. Největší procento aplikací, které pro tento počítač vznikly jsou hry, ovšem existuje i velké množství neherního software.

Přímo v počítači je vestavěno prostředí programovacího jazyka *BASIC*, které by se dalo považovat za obdobu dnešních operačních systémů. Kromě paměti ROM, kde je prostředí BASIC uloženo, však počítač nemá žádné trvalé úložiště pro aplikace. Jediným způsobem (alespoň zpočátku, než byly vyvinuty diskové mechaniky), jak spustit na počítači nějakou aplikaci, bylo buďto ji do paměti počítače nahrát z magnetofonové pásky nebo si ji vlastnoručně naprogramovat. Jen pro představu, nahrání programu z magnetofonové kazety trvá průměrně 2-5 minut (samozřejmě záleží na velikosti).

Dnes již software pro ZX Spectrum není potřeba uchovávat na audiokazetách. Především s rozvojem emulátorů se začaly používat digitální formáty, které buďto ukládají samotný obsah magnetofonové pásky v binární podobě (takovéto formáty se označují

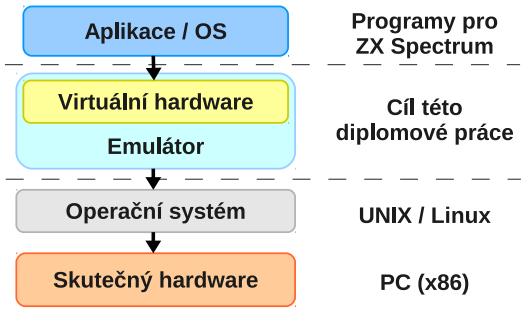
jako *tape image*), nebo ukládají rovnou kompletní snímek obsahu paměti a registrů procesoru počítače (tyto formáty jsou označovány *snapshot*).

Velká část programů pro ZX Spectrum zůstává zachována díky komunitám, jakou je například *World of Spectrum*, která spravuje asi největší online archiv[12] aplikací pro tento dnes již legendární počítač. Většina vývojářů se přitom rozhodla, že umožní šíření svých programů prostřednictvím tohoto archivu bezplatně.

3 Emulace počítače a debugování

Pod pojmem *emulátor* si obecně lze představit software (někdy také hardware), který imituje (neboli emuluje) funkce jednoho systému pomocí jiného rozdílného systému. *Emulátor počítače* pak imituje chování veškerých komponent konkrétního systému. Programy tedy nejsou spouštěny na reálném systému, ale na *virtuálním systému* v prostředí emulátoru. Samotný emulátor již běží na reálném stroji (viz. obrázek 5). Takto je umožněno spouštět programový kód na platformě, pro kterou původně nebyl určen, a to bez nutnosti modifikace samotného kódu.

Za pojmem *debugování* se může skrývat rozsáhlé množství aktivit. Ujasnění tohoto i některých dalších termínů, týkajících se této problematiky, je věnována samostatná podkapitola 3.3. Následující podkapitoly jsou tedy určeny především těm čtenářům, kteří do oblasti emulace počítačů a debugování programů doposud nezavítali.



Obrázek 5: Diagram vrstev při emulaci
(vpravo uvedeny konkrétní realizace vrstev)

3.1 Důvody pro vznik emulátorů

Od poloviny 20. století, kdy začala éra digitálních počítačů, vznikají stále nové počítačové architektury, které nahrazují ty předešlé. Takto každý počítač dříve či později čeká cesta do „křemíkového nebe“[2]. Naproti tomu software svou podstatou – jako posloupnost nul a jedniček – není závislý na konkrétní fyzické reprezentaci. Technický pokrok nás tak staví do pozice, kdy aplikace přetrvávají, ale nemáme již k dispozici odpovídající stroje, na kterých by je bylo možno provozovat.

Díky emulátorům lze i dnes s pomocí běžného osobního počítače spouštět například software určený pro legendární počítač ENIAC[3] aniž bychom vlastnili jeho skutečný exemplář.

[2]Označení pro konec užitelné doby elektronického přístroje. Tento pojem byl popularizován v Britském televizním sci-fi seriálu Red Dwarf.

[3]Historicky první Turingovsky-kompletní elektronický počítač sestavený pro Armádu Spojených států amerických (stavba dokončena v roce 1946).

3.2 Úrovně emulace

Emulaci lze provádět s různými úrovněmi podrobnosti. Pro emulaci počítačů se nejčastěji používá dělení do dvou skupin, kterými jsou nízkoúrovňová emulace – LLE (*Low-Level Emulation*) a vysokoúrovňová emulace – HLE (*High-Level Emulation*).

3.2.1 Nízkoúrovňová emulace

Nízkoúrovňové emulátory se svou vnitřní stavbou drží toho, jak imitovaný systém vypadá ve skutečnosti. Typicky je emulátor rozdělen do několika modulů, které odpovídají komponentám původního systému. Příkladem může být modul pro emulaci CPU, který dekóduje a spouští instrukce strojového kódu emulovaného počítače a uvnitř pracuje s proměnnými, které reprezentují registry emulovaného procesoru.

Do této skupiny se řadí také emulátory, které pracují na úrovni logických hradel či tranzistorů jednotlivých digitálních obvodů. Tyto případy jsou ale píše výjimkou, neboť je ve spoustě případů takřka nemožné získat schémata integrovaných obvodů a emulace na takto nízké úrovni vyžaduje nesmírné množství výpočetního času. Velice zajímavým počinem v této oblasti je projekt Visual 6502, v rámci nějž byl vyvinut emulátor procesoru MOS6502. Tento emulátor pracuje právě na úrovni tranzistorů a navíc nabízí i vizualizaci celého čipu. Emulátor je možné spustit v prohlížeči s podporou technologie HTML5 (viz [11]) a dosahuje rychlosti v řádu jednotek instrukcí za vteřinu. Skutečný procesor přitom zvládá provést během jedné vteřiny i stovky tisíc instrukcí.

Obecně by se dalo říci, že čím je nižší úroveň emulace, tím přesnější je imitace chování systému, avšak za cenu vyšších nároků na výpočetní výkon.

3.2.2 Vysokoúrovňová emulace

Základní myšlenkou vysokoúrovňové emulace je, že se zabýváme tím co komponenty počítače dělají, spíše než tím jak to dělají.

Kupříkladu současné grafické karty používají k vykreslování 3D scén různé typy výpočetních jednotek (GPU), které se diametrálně liší svou vnitřní strukturou. Existuje však tzv. *standardní zobrazovací řetězec* (v anglické literatuře *rendering pipeline*), což je základní schéma, podle kterého probíhá vykreslování 3D scény ve většině grafických karet. Namísto emulace vnitřní logiky čipu grafické karty můžeme vzít popis 3D scény určený pro emulovanou grafickou kartu a tato data převést do formátu, který dokáže vykreslit grafická karta počítače, na kterém emulátor běží. Výsledný obraz nemusí být (a často nebude) naprosto totožný s tím, který by vykreslila skutečná grafická karta, ovšem dosáhneme mnohem vyššího výkonu než v případě nízkoúrovňové emulace.

Pokud mají emulovaný počítač a počítač, na kterém je emulátor spouštěn, srovnatelný výpočetní výkon a usilujeme o emulaci v reálném čase (kdy program v emulátoru poběží stejně rychle jako by běžel na skutečném stroji), pak se použití vysokoúrovňové emulace nevyhneme.

3.3 Debugování – základní pojmy

Pojmem *debugování*[4] (z anglického *debugging*) se obecně chápe proces odstraňování chyb ze systému. Pro debugování software existuje široká řada různých nástrojů a přístupů. Pro účel této práce budeme pod pojmem debugování uvažovat především možnost pozastavení běhu programu (v angličtině *break*) a to buďto ručně nebo pomocí tzv. breakpointů. Během pozastavení budeme očekávat možnost zkoumání obsahu registrů procesoru a také obsahu paměti včetně zobrazení strojového kódu programu v jazyce symbolických instrukcí.

Výraz *breakpoint* se vztahuje k označení místa v kódu programu, na které když se při běhu programu narazí, dojde k pozastavení programu. Samotný breakpoint pro nás tedy bude v podstatě znamenat adresu instrukce strojového kódu programu v paměti počítače.

Strojový kód je přeložený kód *jazyka symbolických instrukcí*[5] (JSI). Zatímco zdrojový kód v jazyce symbolických instrukcí je čitelný text, *strojový kód* jsou binární data, kterým rozumí výpočetní jednotka počítače – procesor.

Pojmem *disassemblování* se označuje proces zpětného převodu ze strojového kódu do zdrojového kódu v jazyce symbolických instrukcí.

[4] Přestože toto slovo nenalezneme ve slovníku spisovné češtiny, v praxi i odborné literatuře se tento výraz hojně používá.

[5] V praxi se často nesprávně používá termín *assembler*. Tento termín však označuje samotný nástroj, který provádí převod z jazyka symbolických instrukcí do strojového kódu.

4 Existující emulátory počítače ZX Spectrum

Tato kapitola je věnována stručnému přehledu některých již existujících emulátorů počítače ZX Spectrum. Je potřeba zdůraznit, že emulace počítače ZX Spectrum není žádnou novinkou. Jedním z prvních emulátorů určených pro IBM PC byl emulátor *NUTRIA*, který vznikl již v roce 1991 (viz [2]). Od té doby bylo vytvořeno veliké množství dalších emulátorů tohoto slavného počítače určených pro všemožné systémy, architektury a zařízení. Už jen podle [13] je jich více než stovka.

Mezi těmito emulátory je široká rozmanitost. Některé jsou určeny pro osobní počítače, jiné třeba pro mobilní telefony, některým k běhu stačí pouze webový prohlížeč. Dále se liší třeba tím, které modely počítače ZX Spectrum (nebo i jiných počítačů) dokáží emulovat anebo které přídavné periférie počítače podporují. Není však mnoho emulátorů, které dokáží nejen počítač emulovat, ale také zkoumat kód spouštěného programu – debuggovat.

Za všechny emulátory byli vybráni čtyři zástupci, u kterých budou v následujících kapitolách vyjmenovány jejich základní rysy a zajímavé vlastnosti. V příloze C jsou pro názornost uvedeny ukázky grafických rozhraní těchto programů.

4.1 Spectaculator

Spectaculator je současnosti jedním z nejvyspělejších emulátorů nejen počítače ZX Spectrum 48K, ale také pozdějších variant jako ZX Spectrum 128, +2, +3 nebo klon Pentagon 128, který byl vyráběn v SSSR. Podporuje emulaci široké škály vstupně-výstupních zařízení od takové samozřejmosti jako je magnetofon, přes tiskárnu až po diskovou jednotku.

Tento emulátor obsahuje také poměrně pokročilý debugger, který zobrazuje disassemblované instrukce a dovoluje editovat obsah paměti a registrů (viz obrázek 18). Dále debugger podporuje krokování a zadávání instrukčních breakpointů.

Aplikace *Spectaculator* je dostupná pouze pro operační systémy Windows (verze XP, Vista a 7) a to buďto jako placená plná verze nebo několikadenní zkušební verze.

4.2 FUSE – Free Unix Spectrum Emulator

Emulátor *FUSE* je multiplatformním open-source emulátorem počítačů řady ZX Spectrum (všechny varianty), Pentagon (128K, 512K a 1024K), Scorpion (ZX 256) a Timex (TC2048, TC2068 a TS2068). Díky otevřenosti kódu do vývoje aplikace přispělo množství lidí, a tak byla do detailů vyladěna přesnost emulace jednotlivých modelů počítačů.

Podporována je spousta vstupně-výstupních zařízení včetně myši *Kempston mouse* nebo různých rozhraní pro práci paměťovými kartami *Compact Flash* a pevnými disky s rozhraním *IDE*.

Emulátor obsahuje také debugger (viz obrázek 19), který zobrazuje disassemblovaný kód v paměti počítače, obsah registrů procesoru a obsah zásobníku. Debugger umí krokovat běh programu a také nastavovat breakpointy. Je potřeba poznamenat, že debugger emulátoru *FUSE* obsahuje příkazovou řádku pro zadávání a vyhodnocování různých

výrazů, avšak k této funkcionalitě nebyla nalezena žádná dokumentace. Nebylo tedy zjištěno zda není alespoň pomocí těchto výrazů možné modifikovat obsah registrů nebo paměti počítače.

Za zmínku také stojí, že při vývoji *FUSE* vznikla samostatná knihovna *libspectrum*, která umožňuje snadnou práci se soubory používanými v emulátoru. Během vývoje emulátoru byla vytvořena řada dalších pomocných programů například pro nahrávání obsahu fyzických magnetofonových pásek do souborů typu *tape image*.

Všechen tento software je volně k dispozici jak pro operační systémy Windows, tak i pro UNIX (Linux, BSD a další).

4.3 jBacteria

Emulátor *jBacteria* jde poněkud jinou cestou než dříve zmíněné emulátory. Jeho hlavní devizou je snadná přístupnost. Jediné, co je pro jeho provoz potřeba je webový prohlížeč s podporou technologie *JavaScript* (viz [5]).

Přestože technologie JavaScript není zrovna vhodná pro aplikace vyžadující vysoký výpočetní výkon (čímž emulace procesoru počítače dozajista je), současné osobní počítače zvládají emulaci v reálném čase.

4.4 TommyGun

Poslední aplikací zmíněnou v této kapitole je vývojové prostředí *TommyGun* (viz [4]). Jedná se o komplexní nástroj, který usnadňuje vývoj především her pro počítač ZX Spectrum i jiné 8bitové a 16bitové počítače. Je určen pro vytváření grafiky, zvukových efektů a hudby. Obsahuje také speciální pomůcky pro návrh jednotlivých úrovní her.

Samotné vývojové prostředí neobsahuje emulátor (viz obrázek 20), ale lze v něm nastavit použití libovolného externího emulátoru, který umožňuje zadat soubor k načtení jako parametr příkazové řádky. Taktéž lze použít externí kompilátor zdrojových kódů z assembleru, Basicu, C nebo C++.

Tento nástroj má veliký potenciál pro vývojáře her pro ZX Spectrum a navíc je šířen jako freeware. Zdrojové kódy však dostupné nejsou a jak se zdá, vývoj stagnoval na poslední verzi 0.9.39 z května roku 2009.

5 Specifikace požadavků

V následujících podkapitolách jsou sepsány požadavky na aplikaci, který má být implementována. Požadavky byly určeny ze zadání a na základě konzultací s vedoucím diplomové práce.

5.1 Funkční požadavky

Funkční požadavky zachycují, čeho má být systém schopen. Tyto požadavky jsou základem pro návrh systému.

5.1.1 Emulace počítače

Ze zadání této práce plyne, že emulátor má umět věrně imitovat počítač ZX Spectrum a to včetně známých, avšak ne oficiálně dokumentovaných, vlastností. Vlastnostmi, které nejsou oficiálně dokumentované se rozumí především operační kódy instrukcí procesoru, které nejsou zmíněny v oficiálním manuálu [19], ale byly zmapovány v [6].

Z analýzy samotného počítače (kapitola 2) dále vyplynuly požadavky na emulaci jednotky ULA. Je potřeba, aby emulátor dokázal vykreslovat obraz, který ve skutečném počítači vytváří jednotka ULA na základě obrazové oblasti paměti RAM. Krom obrazového výstupu ULA zprostředkovává také vstup z klávesnice. Emulátor tedy musí být schopen přijímat stisky kláves na klávesnici počítače, na kterém emulátor běží, a interpretovat je jako stisky kláves v emulovaném počítači.

V počítači je taktéž zvukový výstup. Pomocí vestavěného reproduktoru je přehráván jednobitový zvukový signál generovaný výstupními operacemi procesoru.

5.1.2 Nahrávání programů ze souboru

Emulátor má být schopen načítat soubory s obrazem paměti počítače (*snapshot*) a také soubory s obsahem magnetofonové pásky (*tape image*). To zahrnuje zejména souborové formáty *SNA*, *Z80*, *ZXS* a *SLT* pro soubory typu snapshot a formáty *TAP* a *TZX* pro soubory typu tape image.

Počítač disponuje zvukovým vstupem, který slouží pro nahrávání programů z magnetofonové pásky. Emulátor nahrazuje vstup z magnetofonu soubory typu tape image, takže musí být schopen převést tyto soubory zpět na zvukový signál, který pak bude přiváděn na vstupní port počítače.

5.1.3 Krokování programu

Jedním ze základních požadavků na debugger počítače je možnost krokování programu po jednotlivých instrukcích. Z toho samozřejmě plyne schopnost pozastavit běh emulace na libovolnou dobu. Další požadovanou funkcí, která se řadí do této kategorie, je vytváření instrukčních breakpointů pro účely zastavení emulace před vykonáním instrukce na určité adrese.

5.1.4 Zobrazení obsahu paměti a registrů

Proto, aby bylo umožněno pohodlně debugovat a zkoumat kód programů, je nutné, aby debugger dekódoval operační kódy instrukcí v paměti počítače na zápis v jazyce symbolických instrukcí. Při debugování je také důležité sledovat obsah registrů procesoru. Proto musí debugger zobrazovat hodnotu každého z nich v každém kroku debugování.

5.2 Nefunkční požadavky

Nefunkční požadavky slouží k zachycení nároků na implementovaný systém, které se přímo netýkají jeho funkcí. Jsou to požadavky zaměřené například na výkon nebo uživatelskou přívětivost.

5.2.1 Grafické uživatelské rozhraní

Celou aplikaci musí být možné ovládat pomocí grafického rozhraní. Mělo by se jednat o klasickou desktopovou aplikaci.

5.2.2 Emulace v reálném čase

Emulátor má být schopen emulovat běh počítače v reálném čase. Tento požadavek je kladen poněkud vágně, neboť není přesně definováno jaký výkon bude mít počítač, na kterém bude emulátor spouštěn. Ovšem s ohledem na o několik řádů vyšší výkon současných osobních počítačů oproti počítači ZX Spectrum může být požadavek chápán tak, že program v emulátoru nesmí běžet znatelně pomaleji nebo naopak rychleji než na skutečném počítači.

5.2.3 Cílová platforma

Emulátor je primárně určen pro operační systém Linux. Program však nemá být závislý na konkrétní distribuci systému, nýbrž by se měl držet standardů společných pro většinu operačních systémů typu UNIX.

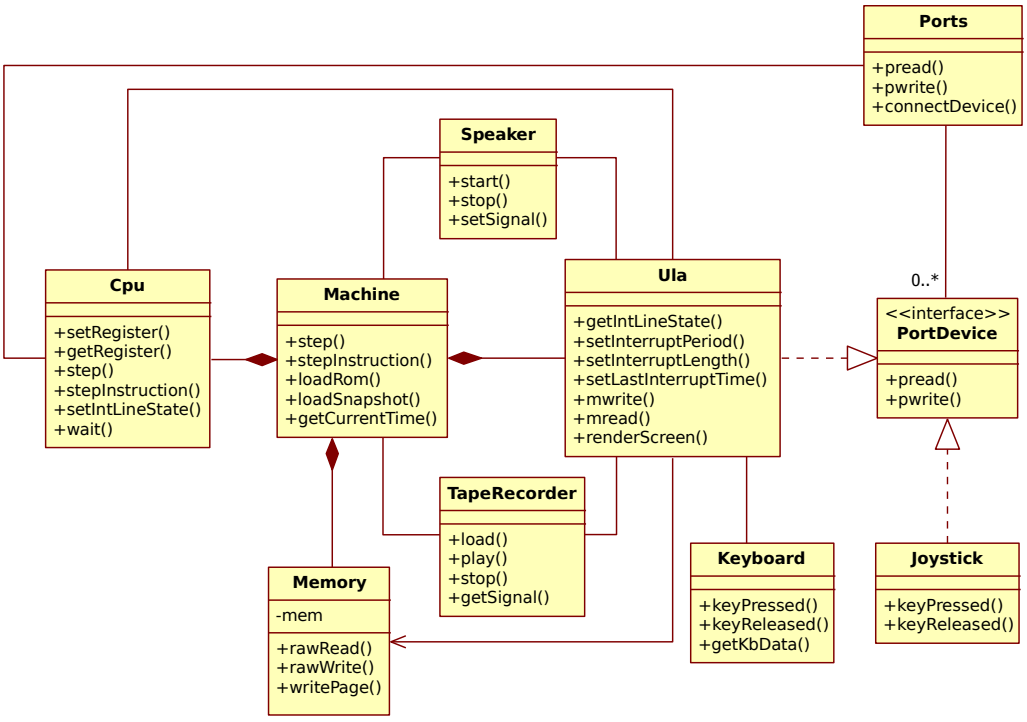
6 Návrh emulátoru

V následujících podkapitolách bude podrobně rozebrána architektura vyvíjeného emulátoru. Návrh se držel zásad objektově-orientovaného programování, takže základem pro vytvoření architektury emulátoru byla objektová dekompozice architektury skutečného počítače. Za popisem statické stránky systému následují dvě kapitoly (6.2 a 6.3), které zachycují zásadní činnosti při běhu v počítače.

6.1 Objektová dekompozice

Návrh architektury emulátoru byl proveden objektovou dekompozicí, tedy tak, že byly identifikovány hlavní komponenty emulovaného počítače, podle nichž byly v systému vytvořeny třídy.

Přehled tříd emulátoru a jejich závislosti znázorňuje diagram tříd na obrázku 6. V diagramu jsou zakresleny všechny třídy, které se účastní na emulaci počítače. Z důvodu přehlednosti zde však není uvedena třída `Emulator`. Její místo v systému a také úlohy ostatních tříd budou vysvětleny v následujících podkapitolách.



Obrázek 6: Diagram tříd emulátoru

6.1.1 Třída Machine

Stěžejní část emulátoru tvoří třída `Machine`, která zapouzdřuje celý počítač a má za úkol zprostředkovat interakci jednotlivých komponent a také omezit jejich vzájemnou závislost podobně jako je to u návrhového vzoru *Mediator*. Ukázalo se však, že vazba mezi procesorem a jednotkou ULA je natolik těsná, že pro zachování jednoduchosti celé architektury je vhodnější, aby mezi sebou třídy `Cpu` a `Ula` měly přímou vazbu.

Třída `Machine` uchovává virtuální čas v rámci emulace. Při každém volání metody `step()` nebo `stepInstruction()` je emulován krátký časový úsek běhu počítače. Aktuální čas v emulátoru lze získat pomocí metody `getCurrentTime()`. Podrobnosti o průběhu emulace jsou k nalezení v kapitole 6.2.

Dále tato třída poskytuje metodu `loadSnapshot()`, která slouží pro načtení stavu celého počítače (především tedy obsah paměti a registrů procesoru) ze souboru typu *snapshot*. Metoda `loadRom()` pak umožňuje nahrát pouze oblast paměti ROM.

6.1.2 Třída Cpu

Třída `Cpu` obstarává samotné vykonávání programového kódu. Pro tento účel poskytuje dvě metody `step()` a `stepInstruction()`. Zatímco druhá ze zmiňovaných metod vždy provede jednu celou instrukci procesoru, metoda `step()` v případě více-bajtových instrukcí zpracuje jen jeden operační kód instrukce.

Během provádění instrukcí může procesor přistupovat do paměti nebo ke vstupně-výstupním zařízením. Přístup do paměti je zprostředkováván třídou `Ula`, konkrétně pak jejími metodami `mread()` a `mwrite()`. Důvodem je především velmi úzká vazba mezi procesorem a jednotkou ULA, která může procesoru v případě přístupu do určitých oblastí paměti dočasně odstavit časovací signál a prodloužit tak dobu provádění instrukce. Toto pozastavení procesoru se provádí voláním metody `wait()` třídy `Cpu`. Více o této problematice v kapitole 6.3.

K provádění vstupně-výstupních operací jsou volány metody `pread()` a `pwrite()` třídy `Ports`, která je popsána v samostatné kapitole 6.1.5.

Metody `setRegister()` a `getRegister()` pak už jen slouží pro přístup k registrům procesoru a metoda `reset()` k uvedení procesoru do stavu, v jakém je po zapnutí počítače.

6.1.3 Třída Ula

V počítači je jednotka ULA jednou z nejdůležitějších komponent a jinak tomu není ani v emulátoru. Třída `Ula` zprostředkovává pro procesor přístup do paměti prostřednictvím metod `mread()` a `mwrite()`.

Jednotka ULA také v počítači generuje požadavky přerušení `INT`. Voláním metody `getIntLineState()` se získá aktuální logická hodnota na výstupu s přerušením a ta je třídou `Machine` přivedena na vstup přerušení třídy `Cpu` (více v kapitole 6.2).

Pro obrazový výstup disponuje třída metodou `renderScreen()`, která do zadaného obrazového bufferu vykreslí obraz, který je u skutečného počítače viditelný na televizní obrazovce.

Třída `Ula` zároveň implementuje rozhraní `PortDevice` (více v kapitole 6.1.5) protože jako vstupně-výstupní zařízení odpovídá na portu 254 (viz též kapitolu 2.6). Přes tento port procesor mimo jiné přistupuje ke klávesnici (třída `Keyboard`), magnetofonu (třída `TapeRecorder`) a vnitřnímu reproduktoru (třída `Speaker`).

6.1.4 Třída Memory

Třída `Memory` ma za úkol uchovávat samotná data v paměti ROM a RAM počítače. Pomocí metody `rawRead()` resp. `rawWrite()` je zajištěno čtení resp. zápis jednotlivých bajtů v paměti. Metoda `writePage()` umožňuje zapsat najednou celé stránky paměti.

6.1.5 Třída Ports a rozhraní PortDevice

Vstupně-výstupní operace (konkrétně pro procesorové instrukce `IN` a `OUT`) zajišťuje třída `Ports`. Tato třída má metody `pread()` a `pwrite()`, které jsou volány z třídy `Cpu`. Při vstupně-výstupním požadavku třída `Ports` postupně předá volání odpovídající metodě všech objektů, které implementují rozhraní `PortDevice` a byly zaregistrovány metodou `Ports::connectDevice()`[6].

Při volání metody `pread()` má každé zařízení možnost modifikovat výsledná data. To odpovídá situaci ve skutečném počítači, kdy jsou všechna zařízení najednou připojena na datovou sběrnici a na základě adresy požadavku buďto přivedou na vodiče datové sběrnice nějaké logické signály, nebo zůstanou jejich výstupy ve stavu vysoké impedance a tak neovlivňují signály ostatních zařízení na sběrnici. Pokud ve skutečném počítači odpoví na požadavek čtení více zařízení najednou, výsledné signály na datové sběrnici nelze předpovědět. V emulátoru jsou data na sběrnici určena vždy poslední zařízením, které na ni zapíše.

6.1.6 Třídy Keyboard a Joystick

Třídy `Keyboard` a `Joystick` si jsou v mnohém podobné. Obě reprezentují vstupní zařízení s tlačítky. Metody `keyPressed()` resp. `keyReleased()` jsou volány zvenčí při stisku resp. uvolněné klávesy na klávesnici či joysticku.

Třída `Joystick` je zařízením implementujícím rozhraní `PortDevice`. Podle toho, která tlačítka joysticku jsou aktuálně stisknutá, generuje přímo binární data, která jsou poslána při vstupním požadavku na datovou sběrnici.

Naproti tomu klávesnice (třída `Keyboard`) je v počítači připojena na datovou sběrnici skrze jednotku ULA. Proto i v emulátoru není klávesnice zařízením typu `PortDevice`, ale data jsou z ní čtena třídou `Ula` prostřednictvím metody `getKbData()`.

[6] Zápisem ve tvaru `A::B()` je označována členská metoda `B` ve třídě `A`.

6.1.7 Třídy TapeRecorder a Speaker

Magnetofon (třída `TapeRecorder`) a vnitřní reproduktor (třída `Speaker`) jsou ve své podstatě zvukovým vstupem resp. výstupem počítače, pro který jednotka ULA provádí analogově-digitální resp. digitálně-analogový jednobitový převod.

Třída `Speaker` pracuje v podstatě jako výstupní buffer, který zaznamenává časový průběh zvukového signálu předávaného ze třídy `Ula` (viz též kapitolu 2.6.3). Metodou `Speaker::setSignal()` třída `Ula` předává výstupní data do bufferu, odkud je tento signál emulátorem již posílán do zvukové karty počítače, na kterém emulátor běží.

Třída `TapeRecorder` má při čtení z pásky naopak úlohu vstupního bufferu. Nejprve je potřeba otevřít soubor s obsahem pásky pomocí metody `load()`. Poté může být spuštěno či zastaveno přehrávání pásky voláním metody `play()` resp. `stop()`. Třída `TapeRecorder` udržuje aktuální pozici pásky, která se při chodu posunuje společně s časem v emulátoru. Voláním metody `getSignal()` je tedy vždy z pásky obdržen signál v aktuálním čase (získaném metodou `Machine::getCurrentTime()`).

6.1.8 Třída Emulator a rozhraní EmulatorListener

Třída `Emulator` stojí v rámci architektury nad všemi dosud zmíněnými třídami. Její hlavní úloha je zajištění běhu emulace v reálném čase. Zatímco tedy třída `Machine` společně s ostatními dříve uvedenými třídami reprezentují stav emulovaného počítače v určitém časovém okamžiku, třída `Emulator` zajišťuje dynamiku systému. Pomocí metod `start()` resp. `stop()` lze spustit resp. pozastavit běh emulátoru v reálném čase.

Zároveň tato třída používá myšlenku návrhového vzoru Observer Synchronization tak, že informuje o událostech, ke kterým během emulace dochází. Posluchači těchto událostí musí implementovat rozhraní `EmulatorListerner` a zaregistrovat se prostřednictvím metody `Emulator::addListener()`. Této konstrukce bude využito při napojení jádra emulátoru na grafické rozhraní (viz také kapitolu 9.1).

Emulátor své posluchače (třídy implementující rozhraní `EmulatorListener`) může informovat o následujících typech událostí:

- *Registers Changed* – došlo ke změně některého z registrů
- *Memory Changed* – došlo ke změně v paměti
- *Emulation Start* – začátek emulace v reálném čase
- *Emulation Stop* – konec emulace v reálném čase
- *Emulation Step* – krok emulace při krokování

Na základě těchto událostí posluchači – budou to komponenty grafického rozhraní – zaktualizují svoji vizuální podobu.

6.2 Spouštění programového kódu

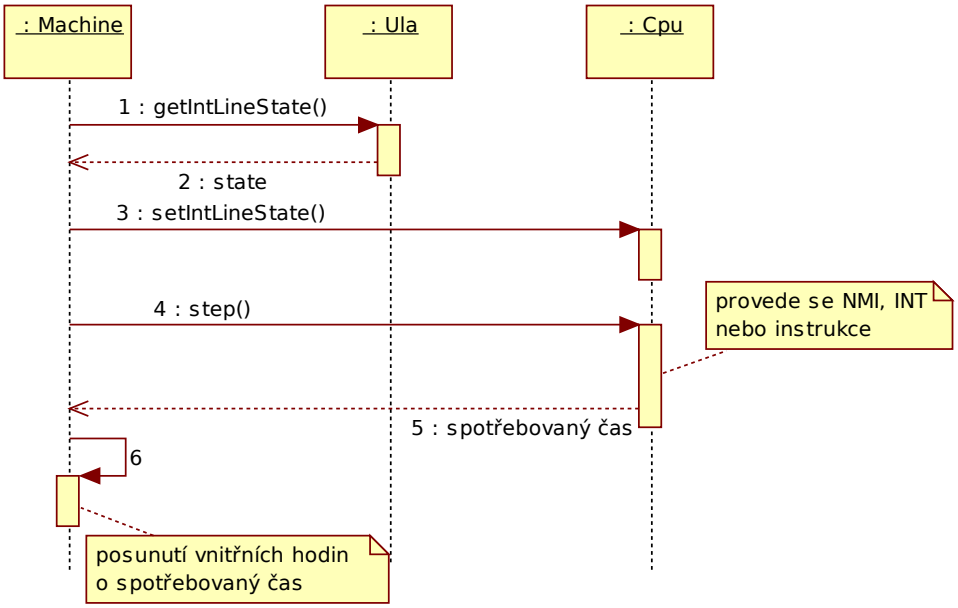
Programový kód aplikací pro počítač ZX Spectrum je v emulátoru spouštěn po krocích voláním metody `step()` třídy `Machine`. Každým zavoláním této metody se vykoná buďto část nebo celá instrukce kódu (případně přerušení) a třída `Machine` zaznamená kolik virtuálního času bylo tímto krokem „spotřebováno“.

Pojem *virtuální čas* představuje dobu, která proběhla v rámci emulace. Odpovídá tomu, jak dlouho by určitá operace trvala na skutečném stroji. Virtuální čas však nemusí být nutně v souladu s časem reálným. Pokud se například rozhodneme emulovat s poloviční rychlostí oproti skutečnosti, proběhne jedna virtuální sekunda za dvě sekundy reálné.

Pokud je emulace prováděna v reálném čase, je metoda `step()` volána z časované smyčky ve třídě `Emulator`. Tato třída zajišťuje, aby bylo prováděné kroky emulace „dávkovány“ v takovém množství, že časové úseky spotřebovaného virtuálního času odpovídají proběhlému reálnému času.

Před každým krokem procesoru třída `Machine` získá hodnotu signálu přerušení `INT` ze třídy `Ula` a předá ji třídě `Cpu`. Třída `Cpu` pak buďto provádí strojový kód, nebo zpracuje přerušení, pokud o něj byla detekována žádost (`INT` nebo `NMI`).

Celý postup jednoho volání metody `Machine::step()` je zakresleno v sekvenčním diagramu na obrázku 7.



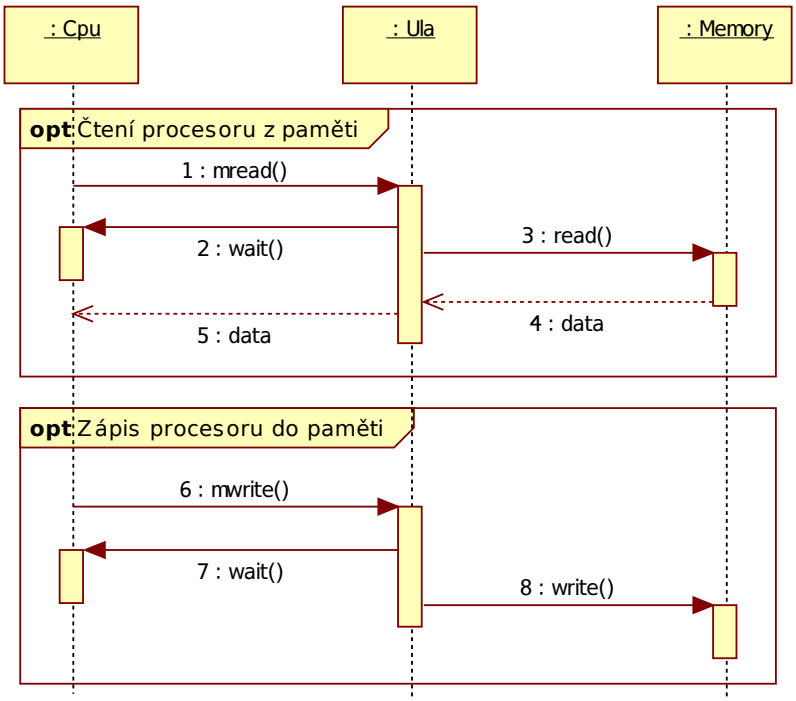
Obrázek 7: Sekvenční diagram kroku emulace počítače

6.3 Přístup CPU do paměti

Třída `Cpu` do paměti přistupuje prostřednictvím třídy `Ula`. Děje se tak proto, že jednotka ULA má v počítači možnost pozastavit hodinový signál CPU při souběžném čtení z videopaměti. Přístup procesoru do obrazové paměti tak může způsobit prodloužení doby potřebné k vykonání instrukce.

Třída `Cpu` tedy zavolá metodu `mread()` nebo `mwrite()` (pro čtení nebo zápis) třídy `Ula` a ta teprve předá požadavek třídě `Memory`, která zapouzdřuje data v paměti ROM a RAM. Přitom může `Ula` zavolat metodu `Cpu::wait()`, která způsobí prodloužení běhu CPU o požadovaný časový interval. Tento postup je zásadní pro přesnou emulaci časování počítače.

Modelové situace přístupu procesoru do paměti během volání metody `Cpu::step()` jsou vyobrazeny v sekvenčním diagramu na obrázku 8.



Obrázek 8: Sekvenční diagram přístupu procesoru do paměti

7 Návrh debuggeru

Z požadavků na debugger vyplynulo, že bude potřeba vytvořit subsystém, který umožní z grafického rozhraní jednoduše přistupovat k registrům a paměti počítače, bude spouštět jednotlivé instrukce (krokování programu) a bude umět převádět strojový kód do jazyka symbolických instrukcí. V následujících podkapitolách jsou popsány třída `Debugger` a rozhraní `DebuggerListener`, které tyto náležitosti zajišťují.

7.1 Třída Debugger

Třída `Debugger` by se dala označit za jakéhosi prostředníka mezi samotným emulátorem počítače a grafickým rozhraním debuggeru. Pomocí metod `setCpuRegister()` a `getCpuRegister()` je umožněn přístup k registrům procesoru. Pro čtení z paměti počítače resp. zápis do ní mohou být použity metody `readMemory()` a `writeMemory()`.

Dále má třída `Debugger` metodu `disassembly()`, která složí k převodu strojového kódu na zadané adrese na textový řetězec se zápisem instrukce v jazyce symbolických instrukcí.

Pro přidávání resp. odebíraní breakpointů slouží metody `addBreakpoint()` resp. `removeBreakpoint()`. Metoda `isBreakpoint()` pak umožňuje zjistit, zda je na dané adrese breakpoint nastaven. Tuto metodu bude využívat emulátor pro zjištění, zda nemá přerušit spouštění programu v reálném čase.

Posledním úkolem třídy je informování posluchačů o změnách stavu debuggeru, konkrétně se jedná o informování o změně množiny breakpointů. Třídy implementující rozhraní `DebuggerListener` se mohou přihlásit k přijmu událostí pomocí metody `Debugger::addListener()`. Takto bude moci například grafické rozhraní transparentně aktualizovat vizuální reprezentaci dat poskytovaných třídou `Debugger`. Více podrobností o rozhraní `DebuggerListener` v následující kapitole 7.2.

7.2 Rozhraní DebuggerListener

Rozhraní `DebuggerListener` musí implementovat každý příjemce událostí z debuggeru. Příjemci se rozumí především komponenty grafického rozhraní, které budou aktualizovat svůj stav podle pravidel daných návrhovým vzorem Observer Synchronization. To znamená, že změny stavu v debuggeru (potažmo emulátoru) jsou třídou `Debugger` propagovány ke komponentám GUI, které pak zobrazí uživateli patřičný výstup. Pokud je některou z komponent grafického rozhraní modifikován stav debuggeru, projeví se tato změna ve všech ostatních komponentách, které jsou posluchači třídy `Debugger` (viz také kapitolu 9.1)

Rozhraní `DebuggerListener` definuje jediný typ události a tou je událost *Breakpoints Changed*, která je vyslána v případě, že došlo ke změně v seznamu breakpointů.

8 Implementace

Tato kapitola je věnována popisu implementace emulátoru a debuggeru. Zdrojové kódy aplikace jsou napsány v programovacím jazyce C++. Veškerý zdrojový kód je k dispozici na přiloženém CD (viz příloha A).

Podkapitoly 8.2-8.11 obsahují podrobnější popis způsobu implementace funkcí systému a také odůvodnění podstatných rozhodnutí, které bylo během implementace aplikace nutno učinit.

8.1 Kód třetích stran

Pro implementaci byly využity některé knihovny třetích stran. Tyto knihovny jsou popsány v následujících podkapitolách 8.1.1-8.1.5. V popise přílohy A je přesně uvedeno, které části zdrojových kódů aplikace jsou původní a které byly převzaty.

8.1.1 Knihovna z80ex

Knihovna *z80ex*[18] je implementací emulátoru procesoru Zilog Z80 v jazyce C. Tato knihovna vznikla původně separací kódu pro emulaci procesoru z emulátoru *FUSE*. Cílem bylo vytvořit samostatnou knihovnu emulující procesor, která bude snadno použitelná v jiných aplikacích a přitom bude přesně emulovat veškeré instrukce procesoru (včetně těch, které nejsou oficiálně zdokumentovány) a také přesně zachová jejich časování.

Součástí knihovny je také podpora disassemblování. Umožňuje tedy převést operační kódy instrukcí procesoru do jazyka symbolických instrukcí.

Knihovna je šířena jako open-source pod licencí GNU GPL verze 2. Knihovna není běžně distribuována v binární podobě, a proto je její zdrojový kód zahrnut v kódu diplomové práce. Použita je knihovna ve verzi 1.1.19.

8.1.2 Knihovna libspectrum

Knihovna *libspectrum*[8] je podpůrnou knihovnou, která má usnadnit práci s nejčastějšími souborovými formáty, které používají emulátory počítače ZX Spectrum. Podporuje jak načítání souborů s obrazem paměti počítače (snapshot), tak i se soubory s obsahem magnetofonové pásky (tape image).

Seznam souborových formátů podporovaných touto knihovnou zahrnuje:

- Snapshot: .z80, .szx, .sna., .zxs, .sp., .snp, +D snapshots
- Tape image: .tzx, .tap, .spc, .sta, .ltp, Warajevo .tap, Z80Em, CSW version 1, .wav

Knihovna je šířena jako open-source pod licencí GNU GPL verze 2 v rámci projektu *FUSE*. Její binární verze je dostupná ve většině nejrozšířenějších distribucích systému Unix. V diplomové práci je použita verze knihovny 1.0.0.

8.1.3 Knihovna PortAudio

Pro možnost zvukového výstupu v emulátoru byla použita knihovna *PortAudio*[9]. Tato multiplatformní open-source knihovna poskytuje poměrně jednoduché rozhraní, které aplikacím zajišťuje jednotný přístup k zařízením pro zpracování zvuku v počítači.

Knihovna je šířena jako open-source pod licencí MIT. Její binární verze je dostupná ve většině nejrozšířenějších distribucích systému Unix. V diplomové práci je použita verze knihovny 19.

8.1.4 Knihovna wxWidgets

Knihovna *wxWidgets*[17] je rozsáhlou multiplatformní knihovnou napsanou v jazyce C++, která poskytuje základní elementy (tzv. *widgety*) pro tvorbu grafických uživatelských rozhraní. Jedním z podstatných rysů této knihovny je, že nemá vlastní grafický vzhled jednotlivých elementů uživatelského rozhraní, ale využívá nativních grafických prvků dané platformy. To znamená, že aplikace bude vždy vypadat tak, jako by byla naprogramována pro grafické rozhraní platformy, na které je spouštěna.

Knihovna je šířena jako open-source pod licencí wxWindows Library Licence. Její binární verze je dostupná jak ve většině nejrozšířenějších distribucích systému Unix, tak i pro systémy Windows. V diplomové práci je použita knihovna ve verzi 2.8.12.

8.1.5 Komponenta HexEditorCtrl

HexEditorCtrl je grafickou komponentou určenou pro knihovnu *wxWidgets*. Není však její standardní součástí, nýbrž vznikla v rámci aplikace *wxHexEditor*[15]. Tento grafický prvek je použit v grafickém rozhraní debuggeru pro hexadecimální zobrazování a editaci obsahu paměti počítače.

Aplikace *wxHexEditor* je dostupná jako open-source pod licencí GNU GPL verze 2. Jelikož je v diplomové práci použita pouze komponenta *HexEditorCtrl*, je distribuována jako součást zdrojových kódů diplomové práce. Zdrojový kód této komponenty pochází z repositáře aplikace[16], konkrétně se jedná o revizi číslo 303.

8.2 Emulace procesoru

Pro emulaci procesoru je využíváno knihovny *z80ex*, která poskytuje nízkoúrovňovou implementaci emulátoru procesoru Zilog Z80. K volbě této knihovny došlo především proto, že za jejími zdrojovými kódy je několik let vývoje, optimalizace a ladění chyb řadou vývojářů. Vytvoření vlastní implementace emulátoru procesoru by s ohledem na množství dokumentovaných i nedokumentovaných vlastností procesoru a časový rozsah práce nevedlo k takové přesnosti emulace, jaké je dosaženo právě použitím této knihovny.

Třída `Cpu` v systému tedy zapouzdřuje funkce poskytované knihovnou *z80ex*. V tabulce 10 je uvedeno, jak si odpovídají jednotlivé metody třídy `Cpu` a funkce knihovny *z80ex*. Funkce `cb_mread()`, `cb_mwrite()`, `cb_pread()` a `cb_pwrite()` jsou tzv. *callback* funkce, což znamená, že implementovány jsou v třídě `Cpu`, ale volány jsou zevnitř knihovny *z80ex*.

Metoda třídy `Cpu`	Funkce knihovny *z80ex*
`getRegister()`	`z80ex_get_reg()`
`setRegister()`	`z80ex_set_reg()`
`mread()` / `mwrite()`	`cb_mread()` / `cb_mwrite()`
`pread()` / `pwrite()`	`cb_pread()` / `cb_pwrite()`
`wait()`	`z80ex_w_states()`

Tabulka 10: Implementace metod třídy Cpu funkcemi knihovny z80ex

Metoda `Cpu::step()` vnitřně pro spouštění instrukcí procesorem volá knihovní funkci `z80ex_step()`. Samotná funkce `z80ex_step()` se však nestará o přerušení procesoru. Požadavky o přerušení jsou detekovány v metodě `Cpu::step()` na základě aktuální a minulé hodnoty na vstupu přerušení `INT` resp. `NMI`. Pokud je požadavek na přerušení detekován, je pro zpracování přerušení zavolána patřičná knihovní funkce `z80ex_int()` či `z80ex_nmi()`.

8.3 Nahrávání obrazu paměti počítače

Načítání souborů typu snapshot, neboli souborů s uloženým stavem počítače, zastává v emulátoru metoda `Machine::loadSnapshot()`. Metodě se předá cesta k souboru na disku a ona již sama načte obsah souboru a předá jej knihovně *libspectrum* prostřednictvím funkce `libspectrum_snap_read()`. Následně se pomocí funkcí s názvem ve tvaru `libspectrum_snap_<registr>()` načtou hodnoty jednotlivých registrů a nastaví se do procesoru metodou `Cpu::setRegister()`.

Obsah paměti je pak načítán po jednotlivých stránkách voláním knihovních funkcí `libspectrum_snap_roms()` a `libspectrum_snap_pages()` a následně zapsán do paměti počítače metodou `Memory::writePage()`. Třída `Machine` má ještě metodu `loadRom()`, která je určena speciálně pro načtení pouze obrazu oblasti paměti ROM. Tato metoda pracuje se vstupním souborem jako se surovými daty.

8.4 Načítání programu z magnetofonu

Třída `TapeRecorder` se stará o načítání programů ze souborů typu tape image. Nejprve je nutné obsah pásky načíst voláním metody `load()`. Tato metoda vnitřně využívá funkci `libspectrum_tape_read()` knihovny *libspectrum*.

Pro získávání obsahu pásky pak slouží metoda `TapeRecorder::getSignal()`. Ta vrací hladinu signálu aktuální pozici pásky. Vnitřně se data čtou knihovní funkcí `libspectrum_tape_get_next_edge()`.

Přehrávání pásky se spouští voláním metody `TapeRecorder::play()`. To zajistí, že je pozice na pásce postupně zvyšována podle času v emulátoru získaného metodou `Machine::getCurrentTime()`. Metodou `TapeRecorder::stop()` lze přehrávání zase zastavit. Voláním metody `TapeRecorder::getPosition()` lze získat aktuální pozici na pásce jako dobu v T-states od začátku pásky.

8.5 Vykreslování obrazu

Obrazový výstup je v emulátoru generován metodou `Ula::renderScreen()`. Tato metoda naplní do zadaného bufferu data jednotlivých bodů obrazu ve formátu RGB. Konkrétně je obraz uložen po řádcích (počínaje levým horním rohem), kdy každý obrazový bod zabírá 32 bitů. V nejnižším bajtu tohoto čísla je uložena modrá barevná složka, poté zelená a ve třetím bajtu červená. Nejvyšší bajt čísla zůstává nevyužit a je nastaven na nulu.

Do bufferu je vykresleno celkem 352x288 obrazových bodů, což odpovídá obrazu o rozlišení 256x192 bodů s okrajem (border) o šířce 48 bodů z každé ze čtyř stran. Zároveň tato metoda obstarává blikání obrazových bloků s příznakem `FLASH` a to tak, že dle aktuálního času v emulátoru střídá při vykreslování s periodou 32 půlsnímků barvu kresby a barvu pozadí.

Na vykreslování obrazu je nejkomplikovanější způsob, jakým je pro určitý bod v obrazovém bufferu emulátoru získáme obrazová data z paměti počítače (viz též kapitolu 2.5). Tento postup je popsán následujícím pseudokódem[7]:

```
for (y = 0 ... 191)
{
   for (x = 0 ... 255)
   {
      tretina = y / 64;
      linka = y mod 8;
      radekVeTretine = (y / 8) mod 8;
      radek = y / 8;
      sloupec = x / 8;

      body_adresa = 0x4000 + (tretina << 11) + (linka << 8) + (radekVeTretine << 5) + sloupec;
      barva_adresa = 0x5B00 + radek << 5 + sloupec;
      ...
      // vypocet barvy bodu
      ...
      buffer[x + 48][y + 48] = barva_bodu
   }
}
```

Metoda `renderScreen()` se volá z metody `EmulatorView::OnPaint()` vždy při překreslení komponenty `panel` v grafickém rozhraní aplikace (GUI aplikace je věnována samostatná kapitola 9).

8.6 Přesné časování paměťových a vstupně-výstupních operací

V kapitole 2.1 bylo zmíněno, že jednotka ULA má možnost pozastavit činnost procesoru odstavením hodinového signálu `CLK` a také se tak děje. Jelikož datová i adresní sběrnice počítače je sdílená mezi jednotkou ULA a procesorem, dochází k prodloužení doby

[7] Poznamenejme, že všechna dělení jsou celočíselná, zápis $a \; mod \; n$ značí zbytek po dělení čísla a číslem n a operace $a << n$ je binárním posuvem bitů čísla a o n bitů vlevo.

spouštění instrukcí procesoru při čtení z paměti nebo přístupu k vstupně-výstupním zařízením.

Implementace této funkcionality je provedena ve třídě `Ula`. Při každém paměťovém požadavku `mread()` nebo `mwrite()` je volána metoda `contendedMemoryDelay()`. Ta na základě adresy v paměti, do které procesor přistupuje a aktuálního času v emulátoru, který uběhl od posledního přerušení `INT`, určí o kolik T-states bude prodlouženo vykonávání aktuální instrukce a následně zavolá metodu `Cpu::wait()`.

Obdobně funguje i zpoždění vstupně-výstupních operací. Pouze s tím rozdílem, že je použita metoda `contendedPortDelay()`, která obstarává výpočet doby zpoždění pro vstupně-výstupní operace. Toto zpoždění se totiž liší od zpoždění operací paměťových.

Objasnění konkrétních podrobností o zpoždění paměťových a vstupně-výstupních operací není náplní této práce. Při implementaci bylo čerpáno z rozsáhlého popisu v oddílech *Contended Memory* a *Contended Input/Output* v [14].

8.7 Emulace v reálném čase

Pro běh emulátoru v reálném čase je zásadní, aby v každém okamžiku čas, který proběhl v rámci emulace odpovídal skutečnému času. Aktuální čas v rámci emulace udržuje třída `Machine` a lze jej získat voláním metody `getCurrentTime()`. Čas, který tato metoda vrací je v jednotkách hodinových cyklů procesoru – T-states. Tato granularita je pro všechny účely dostačující a při použití 64bitového čísla pro uložení tohoto času není reálná nouze, že by byl rozsah čísla přesažen[8].

Jako zdroj reálného času slouží komponenta `wxStopWatch` z knihovny *wxWidgets*. Tato komponenta dokáže zaznamenávat kolik skutečného času uplynulo mezi dvěma voláními jejích metod `Start()` a `Pause()`. Dále je použita komponenta `wxTimer`, která zase umožňuje volání určité metody v pravidelných časových intervalech.

Celá emulace je prováděna „po dávkách“, kdy komponenta `wxTimer` pravidelně s periodou $20\ ms$ volá metodu `Clock::OnTimer()`. V této metodě je zavolána metoda `Emulator::runUntil()`, které je parametrem předán skutečný čas v milisekundách získaný z metody `Clock::getTime()`. Metoda `runUntil()` pak ve smyčce provádí kroky emulace počítače voláním `Machine::step()`, dokud aktuální čas v rámci emulátoru nedorovná zadaný skutečný čas.

Perioda $20\ ms$ pro dávkování emulace byla zvolena na základě toho, že je dostatečná z hlediska odezvy na uživatelské vstupy z klávesnice nebo joysticku. Zároveň je to přesně perioda, v jaké probíhá na skutečném počítači vykreslování obrazových snímků (resp. půlsnímků) na televizní obrazovku.

8.8 Přehrávání zvuku

Přehrávání zvuku v emulátoru obstarává třída `Speaker`. V počítači ZX Spectrum je vestavěn reproduktor, který vydává zvuk generovaný změnami bitu 3 a 4 výstupního portu č. 254 (detailněji v kapitole 2.6.3).

[8] Číslo o rozsahu 64bitů dokáže pro 3,5 milionu cyklů procesoru za vteřinu ($3,5\ MHz$) zachytit časový rozsah několika desítek tisíc let.

V emulátoru zachycuje výstupní požadavky na port 254 třída `Ula` a předává zvukový signál třídě `Speaker` prostřednictvím metody `Speaker::setSignal()`. Tyto hodnoty si třída `Speaker` (společně s časem, ve který nastaly) ukládá do bufferu `soundBuffer` k následnému zpracování.

Pro samotný zvukový výstup na zvukovou kartu emulátoru slouží knihovna *PortAudio*. Prostřednictvím funkce `Pa_OpenStream()` se nejprve otevře výstupní kanál pro přehrávání zvuku. Přitom se zaregistruje metoda `Speaker::paCallback()` jako obslužná rutina pro naplňování získání zvukových snímků pro zvukovou kartu. Po zavolání funkce `Pa_StartStream()` začne knihovna v novém vlákně v poměrně rychlém sledu volat metodu `Speaker::paCallback()`, která vždy na základě úrovní signálů dříve uložených v bufferu `soundBuffer` plnit zvukové snímky do výstupního bufferu `outputBuffer` knihovny *PortAudio*.

Kvůli způsobu jakým je prováděna emulace v reálném čase (viz kapitola 8.7) je nutné zavést u zvuku jisté zpoždění. Běh emulátoru probíhá po dávkách o délce $20\ ms$, takže i zvuková data jsou dostupná po dávkách vždy zpětně za $20\ ms$. Aby byla zvuková data dostupná průběžně, je nutné přehrávat zvuk opožděný alespoň o délku dávky zpracování, tedy $20\ ms$. Během implementace bylo však zjištěno, že pro plynulou reprodukci zvuku je potřeba nastavit zpoždění zvuku až na hodnotu okolo 500 milisekund. Ani podrobným laděním se nepodařilo zjistit, která část systému je zdrojem tohoto zpoždění.

Jelikož by ke zvukovému zvukovým bufferu `soundBuffer` mohlo přistupovat více vláken najednou, což je nežádoucí, je pro zamezení souběhu zavedeno vzájemné vyloučení pomocí struktury `pthread_mutex` ze standardní knihovny *pthreads*. Zamykání mutexu se provádí v metodách `setSignal()` a `paCallback()` třídy `Speaker`.

8.9 Emulace joysticku

Emulace joysticku pracuje na dvou úrovních. V rámci třídy `Emulator` je vytvořena instance třídy `Joystick`, která implementuje rozhraní `PortDevice`. Tento objekt je připojen k počítači metodou `Ports::connectDevice()`.

Pro zachytávání stisků kláves ze skutečného joysticku (připojenému k počítači, na kterém běží emulátor) je použita komponenta `wxJoystick`. Tato komponenta je volitelnou součástí knihovny *wxWidgets*. Zda je knihovna *wxWidgets* zkompilována s podporou joysticku je zjišťováno podle definice makra `wxUSE_JOYSTICK`.

Komponenta `wxJoystick` nabízí přístup k získávání uživatelského vstupu pomocí událostí typu `wxJoystickEvent`, které jsou buďto pravidelně odesílány v přednastaveném intervalu nebo je událost poslány vždy, když je stisknuto tlačítko joysticku či se hodnota některé z analogových os změní o více než určitou přednastavenou mez.

Bohužel se při implementaci ukázalo, že použitá verze 2.8.12 knihovny *wxWidgets* nejspíše kvůli chybné implementaci na platformě Linux neodesílá události o pohybu analogovými osami joysticku. Proto byl zvolen alternativní přístup a namísto zachytávání událostí dochází k pravidelnému dotazování na stav tlačítek a os joysticku v metodě `EmulatorView::OnJoystickTimer()`. Metoda je volána s periodou 10 milisekund z vlákna časovače `joystickTimer` (člen třídy `EmulatorView`).

Zatímco joystick Sinclair má pouze tlačítka, které poskytují výstup v podobě dvou hodnot (stlačeno – nestlačeno), komponenta `wxJoystick` používá pro ovládání pohybu analogové vstupy (a to i v případě, že je k počítači připojen joystick či gamepad bez analogových os). Proto se v metodě `EmulatorView::OnJoystickTimer()` provádí převod analogové hodnoty na dvouhodnotovou logiku. V případě, že se hodnota analogová hodnota pohybuje mezi polovinou maximální hodnoty a středové (klidové) hodnoty osy, není tlačítko považováno za stlačené. Když hodnota osy přesáhne polovinu svého maxima (krajní hodnotu lze zjistit pomocí metody `wxJoystick::GetXMax()`) směrem od centrální polohy, je tlačítko považováno za stlačené.

8.10 Disassemblování strojového kódu

Převod strojového kódu procesoru Zilog Z80 do jazyka symbolických instrukcí je zajišťován metodou `Debugger::disassembly()`. Tato metoda vnitřně využívá funkci `z80ex_dasm()` z knihovny *z80ex*.

Tato funkce, na základě zadané adresy v paměti, vrátí řetězec se zápisem instrukce umístěné na této adrese v jazyce symbolických instrukcí. Zároveň je vrácena také délka (v bajtech) této instrukce, takže je možné postupně od určité počáteční adresy přeložit celý obsah paměti.

8.11 Lokalizace

Aplikace byla vytvořena s podporou lokalizace. Konkrétněji je tím myšlen překlad všech textových řetězců používaných v grafickém rozhraní. Lokalizace je zajištěna systémem *gettext*, který je v knihovně grafického rozhraní *wxWidgets* implementován komponentou `wxLocale`.

V emulátoru je ještě před inicializací hlavního okna aplikace (třída `EmulatorView`) volána funkce `InitLanguageSupport()`, která zajistí načtení řetězců ve výchozím jazyce uživatele, který aplikaci spustil. Pakliže není k dispozici překlad v daném jazyce, jsou použity originální textové řetězce ze zdrojových kódů (v tomto případě je to angličtina).

Ve zdrojových kódech jsou veškeré řetězce určené k překladu uvozeny speciálním znakem *podtržítko*, což je makro, které zajišťuje, že se řetězec za běhu aplikace vloží v požadovaném jazyce. Například použití zápisu `_("Open file")` způsobí, že bude celý výraz vyhodnocen jako řetězec `"Otevřít soubor"`.

Pro aplikaci aktuálně existují překlady do dvou jazyků – češtiny a angličtiny. Zdrojové soubory s přeloženými řetězci mají příponu `.po` se nachází ve složce `po` v kořenovém adresáři se zdrojovými kódy aplikace. Po překladu aplikace se ze souborů `.po` vytvoří jejich binární podoba s příponou `.mo`. Tyto soubory jsou pak umístěny pod následující cestou: `lang/<kód jazyka>/LC_MESSAGES/sus107-dt.mo`.

9 Grafické rozhraní

Grafické uživatelské rozhraní aplikace postaveno nad systémem *wxWidgets* (viz také kapitolu 8.1.4). Návrh grafického rozhraní byl vytvořen pomocí nástroje *wxGlade*. Tento nástroj podporuje poměrně širokou sadu prvků grafického rozhraní, jako jsou vstupní pole, tlačítka, menu, seznamy apod. Jednotlivé komponenty se skládají a vytváří se z nich rozložení okna. Nástroj umožňuje nastavit výchozí vlastnosti komponent a také zadat názvy metod zpracujících události, které komponenty mohou vyvolat. Z celého návrhu lze nakonec přímo vygenerovat zdrojové kódy v jazyce C++.

Soubor s návrhem grafického rozhraní v nástroji *wxGlade* je k nalezení pod cestou `ui/sus107-dt.wxg` v kořenové složce se zdrojovými soubory aplikace. Vygenerované soubory v jazyce C++ jsou pak uloženy ve složce `src/ui`. Vygenerované soubory slouží jako jakási šablona, kterou je možno dále upravovat. Nástroj *wxGlade* má však v souborech své speciální bloky kódu, které by neměly být modifikovány, protože při přegenerování budou přepsány. Příklad takovéhoto bloku je uveden v následujícím výpisu:

```
// begin wxGlade: EmulatorView::methods
    void set_properties();
    void do_layout();
// end wxGlade
```

Aplikace sestává celkem ze tří oken. Základním oknem emulátoru, které je otevřeno po spuštění aplikace, je okno `EmulatorView`. Další dvě okna náleží debuggeru. První z nich – `DebuggerView` – je hlavní okno debuggeru se kódu v jazyce symbolických instrukcí a hexadecimálního editoru obsahu paměti. Posledním oknem debuggeru je okno `DebuggerRegistersView`, které nabízí náhled na registry procesoru.

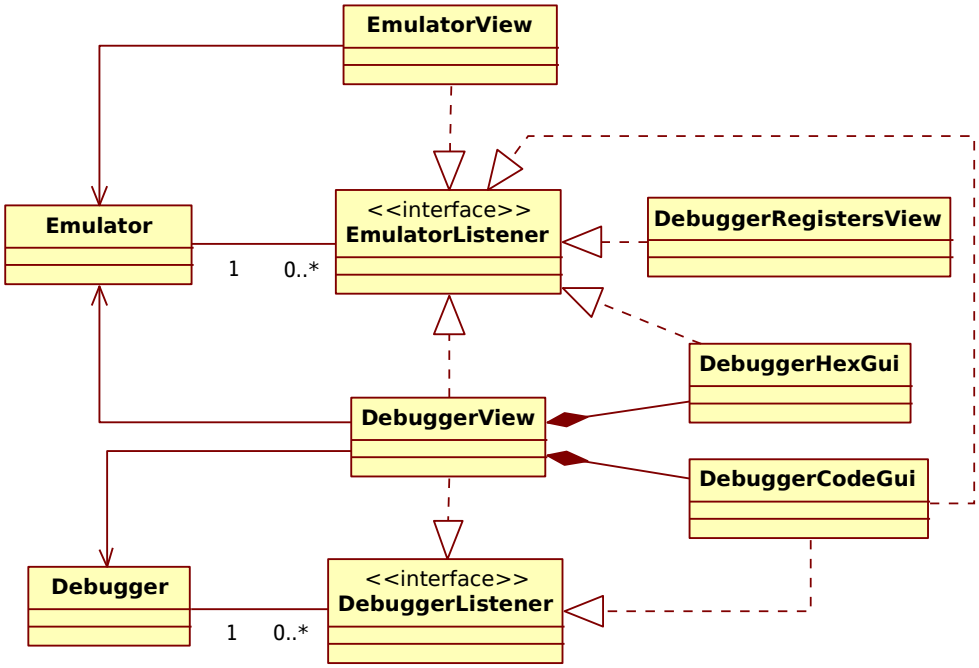
V následujících podkapitolách budou podrobněji popsána jednotlivá okna. Budou tedy vyjmenovány a jejich hlavní ovládací prvky a také bude uvedeno odkud jsou čerpána zobrazovaná data a jaké akce jsou napojeny na události zasílané komponentami. Nejprve se ale v kapitole zmíníme o základním principu, jakým jsou okna a komponenty informovány o změnách dat, která zobrazují.

9.1 Návrhový vzor Observer Synchronization

Pokud požadujeme, aby několik oken v grafickém rozhraní poskytovalo vizuální podobu jednoho *modelu* (sdílených dat), je ideálním řešením využití návrhového vzoru *Observer Synchronization*. Základní myšlenkou tohoto návrhového vzoru je, že každé z oken je zaregistrováno jako posluchač modelu a zároveň může model modifikovat. V případě modifikace pak model rozešle informaci o změně svých dat všem posluchačům a takto se tato změna propaguje do všech oken.

Konkrétně v emulátoru jsou modely třídy `Emulator` a `Debugger`. Posluchači jsou pak okna nebo jednotlivé komponenty grafického rozhraní, které implementují rozhraní `EmulatorListener` nebo `DebuggerListener`.

Na třídním diagramu na obrázku 9 je znázorněna hierarchie a vazby mezi třídami, které se účastní zobrazování grafického rozhraní. Pro přehlednost jsou záměrně v dia-



Obrázek 9: Diagram tříd komunikace s GUI

gramu vynechány jednostranné asociace ze tříd `DebuggerHexGui`, `DebuggerCodeGui` a `DebuggerRegistersView` na třídy `Emulator` a `Debugger`.

9.2 Okno EmulatorView

Okno `EmulatorView` je hlavním oknem emulátoru (viz obrázek 10). Okno reaguje na vstupy z klávesnice pomocí metod `OnKeyDown()` a `OnKeyUp()`. Písmena jsou mezi skutečnou klávesnicí a virtuální klávesnicí v emulátoru mapovány 1:1. Stejně tak klávesa *ENTER*. Číselné klávesy lze zadávat pomocí numerické klívesnice. Klávesa *CAPS SHIFT* počítače ZX Spectrum je na skutečné klávesnici mapována na klávesu *SHIFT*, klávesa *BREAK* na mezerník a klávesa *SYMBOL SHIFT* na klávesu *CTRL*. Hlavní okno dále zpracovává události joysticku.

Třída `EmulatorView` je posluchačem událostí rozhraní `EmulatorListener`. Na základě událostí *Emulation Start* a *Emulation Stop* nastavuje položkám v menu, zda mají býti aktivní.

Menu okna emulátoru je rozvrženo následovně:

- *Soubor*
  - *Otevřít...* – načtení souboru typu snapshot
  - *Konec* – ukončení aplikace
- *Počítač*

  - *Spustit* – spuštění emulace v reálném čase
  - *Zastavit* – zastavení emulace v reálném čase
  - *Resetovat* – resetování počítače
  - *Debugovat...* – spuštění debuggeru (`DebuggerView`)
- *Magnetofon*
  - *Otevřít...* – otevření souboru typu tape image
  - *Přehrát* – spuštění kazety[9].
  - *Zastavit* – zastavení kazety
  - *Přetočit* – přetočení kazety na začátek

Obrázek 10: Diagram tříd komunikace s GUI

9.3 Okno DebuggerView

Okno `DebuggerView` je hlavním oknem debuggeru. Střední část okna je rozdělena na dvě záložky – *Kód* (`DebuggerCodeGui`) a *Hexadecimální* `DebuggerHexGui` – které budou popsány v následujících podkapitolách.

[9]Pro nahrání obsahu pásky do počítače je nutné v počítači v rozhraní BASIC zadat příkaz `LOAD ""`. V závislosti na požité ROM je tento příkaz možné zapsat buďto jako sekvenci stisků kláves `J`, `CTRL+P`, `CTRL+P`, `ENTER`, nebo jako `L`, `O`, `A`, `D`, *mezerník*, `CTRL+P`, `CTRL+P`, `ENTER`.

Podobně jako u okna `EmulatorView` i třída `DebuggerView` naslouchá událostem rozhraní `EmulatorListener` a podle nich nastavuje položkám v menu a nástrojové liště, zda jsou aktivní či nikoliv.

V horní části okna je menu a pod ním nástrojová lišta. Položky menu jsou následující:

- *Úpravy*
  - *Jít na adresu...* – otevře dialog s možností zadání adresy pro přesun kurzoru
- *Zobrazit*
  - *Registry...* – zobrazí okno *Registry* (`DebuggerRegistersView`)
- *Debugger*
  - *Pokračovat* – spuštění emulace v reálném čase
  - *Pozastavit* – zastavení emulace v reálném čase
  - *Instrukční krok* – provedení jedné instrukce

Nástrojová lišta obsahuje v podstatě totožné akce, až na tlačítko s nápisem „PC“, které posouvá kurzor na adresu aktuálně prováděné instrukce dle registru `PC`.

9.3.1 Komponenta DebuggerCodeGui

Komponenta `DebuggerCodeGui` se nachází v první záložce hlavního okna debuggeru (viz obrázek 11). Tato komponenta slouží pro zobrazení obsahu paměti přeloženého do jazyka symbolických instrukcí.

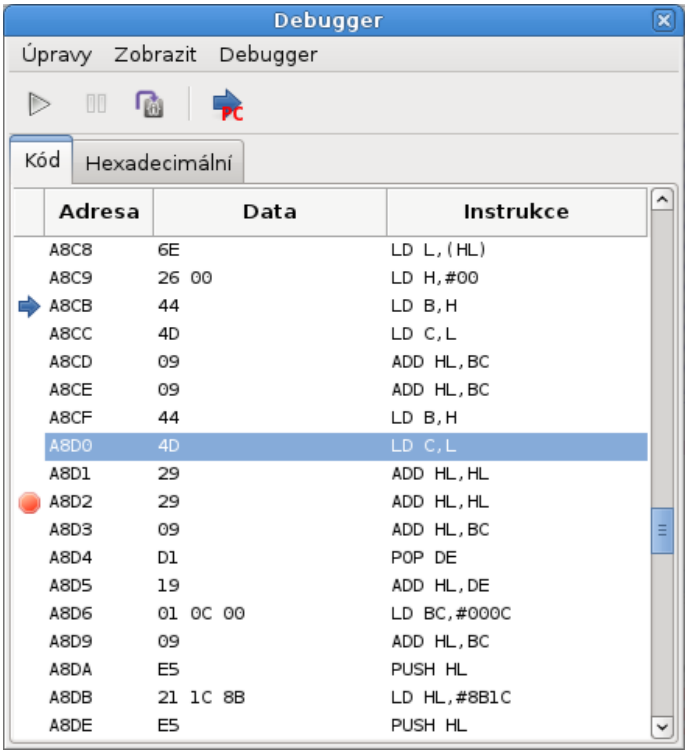
Komponenta implementuje jednak rozhraní `EmulatorListener`, ale také rozhraní `DebuggerListener`. Z událostí rozhraní `EmulatorListener` reaguje na všechny typy událostí, aby došlo k aktualizaci jak při změně registrů (kvůli `PC`) a při změně paměti, tak i při spuštění nebo zastavení emulace a instrukčním kroku. Komponenta naslouchá také události *Breakpoints Changed* rozhraní `EmulatorListener`, kvůli zobrazování ikon na adresách označených breakpointem.

Dvojitým kliknutím na obsah buňky ve sloupci data je možné editovat hexadecimální zápis instrukce. Dvojité kliknutí do místa vlevo od adresy instrukce přepne (přidá nebo odstraní) instrukční breakpoint na dané adrese. Stejnou akci lze provést přes položku *Přepnout breakpoint* v kontextovém menu, které se otevírá kliknutím pravým tlačítkem.

Pomocí druhé volby kontextového menu – *Označit v hexadecimálním zobrazení* – dojde k označení bajtů v záložce *Hexadecimální*, které náleží aktuálně označeným instrukcím kódu.

9.3.2 Komponenta DebuggerHexGui

Komponenta `DebuggerHexGui` se nachází ve druhé záložce okna debuggeru. Tato komponenta slouží pro editaci hexadecimálních dat v paměti počítače. Naslouchá přitom



Obrázek 11: Okno debuggeru se zobrazenou záložkou DebuggerCodeGui

pouze událostem *Memory Changed*, *Emulation Stop* a *Emulation Step*. Tato komponenta je založena na komponentě třetí strany `HexEditorCtrl` (viz kapitola 8.1.5).

V kontextovém menu, aktivovaným pravým tlačítkem myši, jsou k nalezení akce pro zkopírování označených dat do schránky a vkládání ze schránky. Pokud je při vkládání vybrána určitá oblast a ta je kratší než data ve schránce, při vložení se přepíše obsahem stránky jen označená oblast.

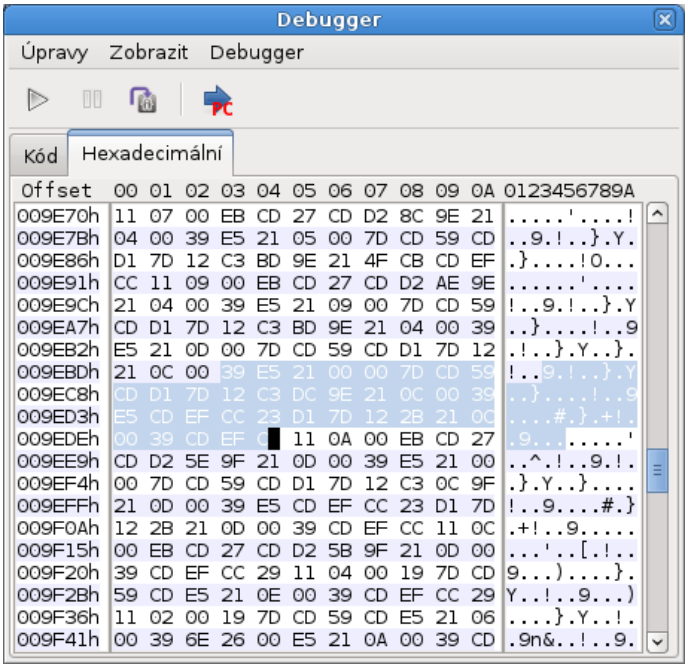
Další funkcí kontextového menu je export vybrané části dat do souboru. Výběrem položky *Uložit jako soubor...* je uživatel vyzván k zadání cílového souboru, do kterého mají být data uložena.

Poslední položkou kontextového menu je možné označit vybrané adresy jako instrukce v záložce *Kód* (`DebuggerCodeView`).

9.4 Okno DebuggerRegistersView

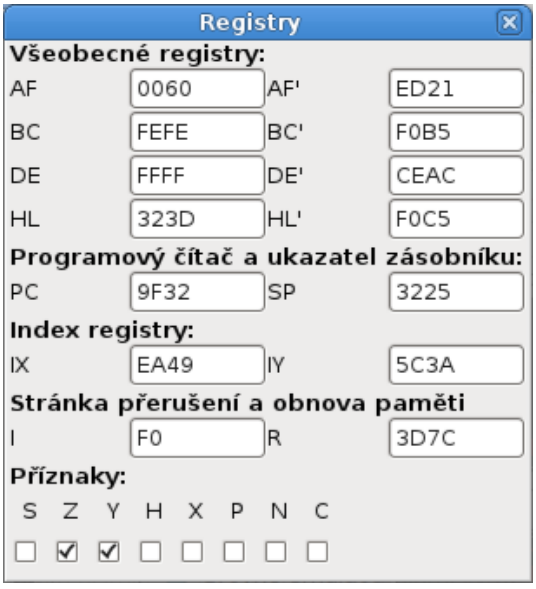
Okno `DebuggerRegistersView` slouží k manipulaci s registry procesoru. Do vstupních polí je možné zadávat hexadecimální hodnoty a potvrzením klávesou ENTER dojde k uložení modifikované hodnoty do registru procesoru.

Toto okno tedy naslouchá událostem *Registers Chnaged*, *Emulator Stop* a *Emulator Step* rozhraní `EmulatorListener` pro účely aktualizace zobrazení registrů.



Obrázek 12: Okno debuggeru se zobrazenou záložkou DebuggerHexGui

Spodní část okna (viz obrázek 13) zobrazuje registr příznaků jako jednotlivé bity, jejichž hodnotu lze kliknutím změnit.



Obrázek 13: Okno debuggeru s obsahem registrů procesoru

10 Testování

Pro ověřování funkčnosti emulátoru byly prováděny dva druhy testů. První kategorií byly testy subjektivní, které sloužily k odhalení snadno viditelných chyb. Druhou kategorií byly testy pomocí speciálních programů pro počítač ZX Spectrum, které mají za úkol důkladně prověřit přesnost emulace.

10.1 Subjektivní testování

Pro testování emulátoru byly prováděny subjektivní uživatelské testy, kdy bylo náhodně vybráno několik programů, které byly postupně spouštěny v emulátoru a na skutečném exempláři počítače ZX Spectrum 48K. Tyto testy pomohly odhalit mimo jiné to, že programy v emulátoru běží přibližně o 7% rychleji než na skutečném stroji. Tato odchylka je připisována nedokonalé emulaci zpoždění procesoru při přístupu do paměti a provádění vstupně-výstupních operací (viz kapitola 8.6). Výsledky testů speciálními aplikacemi tuto domněnku částečně potvrzují (viz následující kapitolu 10.2).

Další zjevný rozdíl mezi emulátorem skutečným počítačem byl objeven při nahrávání programů z magnetofonové pásky. Zatímco u skutečného počítače lze během nahrávání z pásky na televizní obrazovce sledovat na okrajích pohybující se barevné pruhy, v emulátoru okraj pouze poblikává. Efekt barevných pruhů na obrazovce televizoru je způsoben tím, že při načítání dat z pásky dochází k velmi rychlému přepínání barev okraje obrazovky. Toto přepínání je tak rychlé, že paprsek vykreslující obraz v televizoru stačí za tuto dobu uběhnout jen několik řádků. Proto jsou na obrazovce viditelné tyto pruhy. Naproti tomu v emulátoru je vykreslování okraje obrazovky prováděno najednou jedinou barvou.

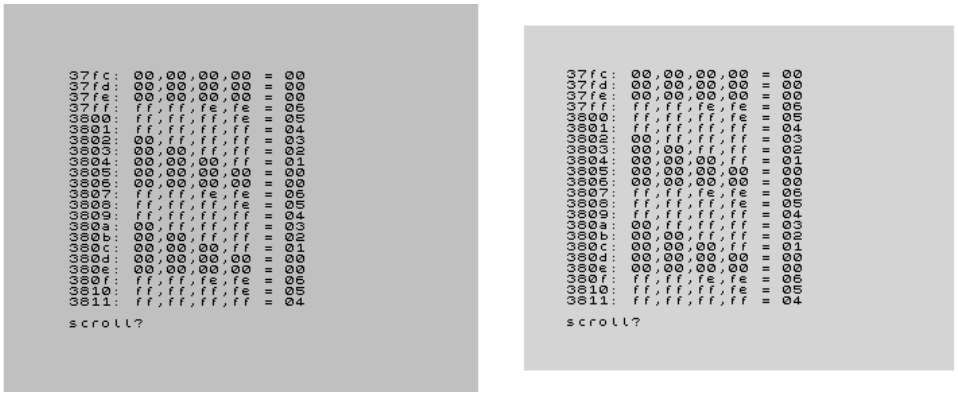
10.2 Speciální testovací aplikace

Pro podrobné testování přesnosti emulace počítače ZX Spectrum existují speciální testovací programy. Na webové stránce [1] jsou dostupné některé z nich společně s online referenční implementací emulátoru počítače.

Byly provedeny testy *fusetest*, *contention* a *iocontention*. Výsledky těchto testů jsou zachyceny na obrázcích v následujících podkapitolách. Vlevo je vždy výstup na obrazovce emulátoru, který byl vytvořen v rámci diplomové práce a vpravo je pro srovnání snímek z referenčního emulátoru *Qaop*.

10.2.1 Test contention

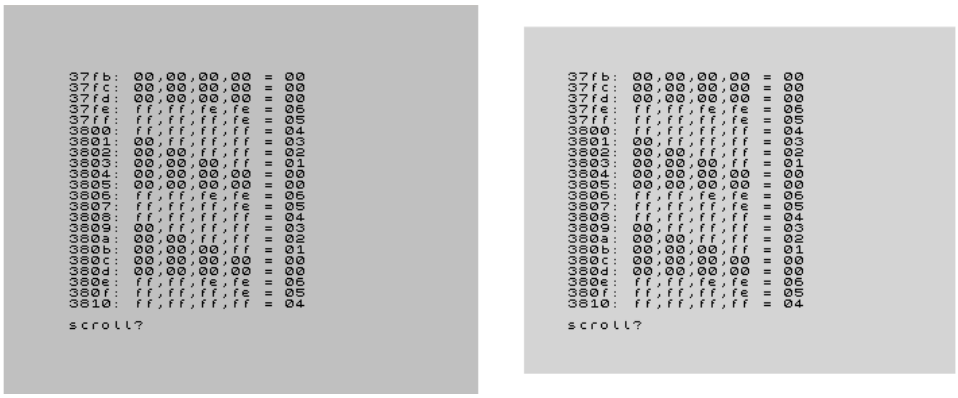
Tento test má za úkol prověřit správnou emulaci zpoždění procesoru vlivem souběžného přístupu do paměti procesorem a jednotkou ULA. Podle shody obsahu obou obrazovek na obrázku 14 je patrné, že test proběhl stejně jak na testovaném emulátoru (vlevo), tak na emulátoru referenčním (vpravo).



Obrázek 14: Výsledky testu contention

10.2.2 Test iocontention

Test *iocontention* má prověřit správnou emulaci zpoždění procesoru vlivem souběžných vstupně-výstupních operací procesoru a jednotky ULA. Z obrázku 15 je patrné, že rovněž druhý test proběhl úspěšně, neboť testovací výstupy na testovaném emulátoru (vlevo) a na emulátoru referenčním (vpravo) jsou totožné.



Obrázek 15: Výsledky testu iocontention

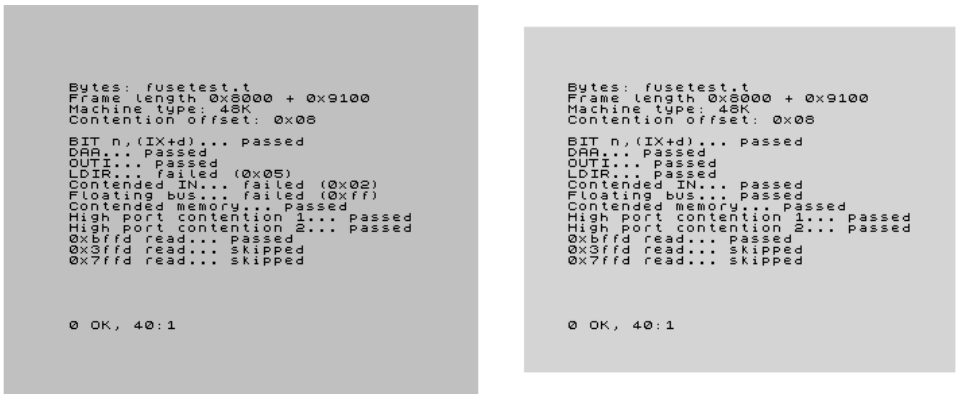
10.2.3 Test fusetest

Poslední test – *fusetest* – se také soustředí na zpoždění procesoru jednotkou ULA, ale provádí testy pomocí více druhů procesorových instrukcí. Z obrázku 16 je možné vyčíst, že emulátor z diplomové práce neprošel následujícími kroky testu:

```
LDIR... failed (0x05)
Contended IN... failed (0x02)
Floating bus... failed (0xff)
```

Test *LDIR* prověřuje zpoždění procesoru při provádění instrukce blokového přesunu dat v paměti `LDIR`. Test *Contended IN* testuje zase zpoždění při vstupní operaci `IN`.

Poslední neúspěšný test – *Floating bus* – testuje nedokumentované chování při čtení z portu, na kterém neodpovídá žádné vstupně-výstupní zařízení. V takovémto případě by měla zůstat datová sběrnice počítače ve stavu, kdy jsou na všech vodičích hodnoty bitů jedna. Současně se vstupní operací `IN` však může jednotka ULA na druhé oddělené části sběrnice číst data z obrazové paměti. Tyto signály se pak přenesou i na druhou polovinu sběrnice, kde je procesor přečte namísto samých jedniček.



Obrázek 16: Výsledky testu fusetest

11 Závěr

Při zpracovávání diplomové práce byla podrobně prozkoumána architektura domácího 8bitového počítače ZX Spectrum 48K. Ukázalo se, že ačkoliv se na první pohled jedná o poměrně jednoduchý systém, je pro jeho dokonalé pochopení potřeba prostudovat spoustu detailních informací o jeho hardwaru.

Byly zmapovány existující emulátory tohoto počítače a na základě získaných poznatků byl navržen a implementován vlastní emulátor. Cílem přitom nebylo soupeřit s ostatními emulátory v počtu podporovaných modelů počítačů nebo přesnosti jejich emulace. V těchto kategoriích již v podstatě došlo k jakémusi nasycení, kdy jsou dostupné emulátory, které imitují veškeré počítače s architekturou více či méně podobnou počítači ZX Spectrum a to dokonce velmi věrohodně. Proto bylo hledáno slabé místo těchto emulátorů a tím byl většinou debugger, který buďto nebyl dostupný vůbec, nebo neposkytoval rozšířené funkce jako například možnost editace kódu v paměti počítače.

Výsledkem se stal emulátor, který kromě imitace samotného počítače ZX Spectrum 48K poskytuje také velice užitečný nástroj v podobě integrovaného debuggeru, který umožňuje zkoumat taje kódů programů, vytvořených mnohdy před několika desítkami let.

Přestože emulátor v současnosti dokáže imitovat pouze jediný model počítače, architektura emulátoru je navržena poměrně robustně a s ohledem na nepříliš velké rozdíly ostatních modelů počítačů, by nemělo být složité jej dále rozšiřovat.

12 Literatura

[1] BOBROWSKI, Jan. Speccy tests. *Qaop ZX Spectrum emulator* [online]. [cit. 2012-04-30]. Dostupné z: http://wizard.ae.krakow.pl/ jb/qaop/tests.html

[2] FERNÁNDEZ-MADRIGAL, Juan-Antonio. *NUTRIA*: one of the (few) oldest ZX Spectrum emulators [online]. © 2002-2004 [cit. 2012-04-27]. Dostupné z: http://jafma.net/software/nutria/

[3] OWEN, Andrew a Chris SMITH. OpenSE BASIC. *SourceForge.net* [online]. 2012-04-29 [cit. 2012-05-02]. Dostupné z: http://sourceforge.net/projects/sebasic/

[4] THOMPSON, Tony. *TommyGun: a retro development toolkit* [online]. © 2005-2009 [cit. 2012-04-28]. Dostupné z: http://www.users.on.net/ tonyt73/TommyGun/

[5] VILLENA, Antonio. *jBacteria: the smallest javascript spectrum emulator* [online]. [cit. 2012-04-28]. Dostupné z: http://jbacteria.retrolandia.net/

[6] YOUNG, Sean. *The Undocumented Z80 Documented*. 2005. Dostupné z: http://www.z80.info/zip/z80cpu_um.pdf

[7] How many Commodore 64 computers were really sold?. In: Pagetable.com: Some Assembly Required [online]. 1. únor 2011 [cit. 2012-03-25]. Dostupné z: http://www.pagetable.com/?p=547

[8] Libspectrum - emulator support library. *SourceForge.net* [online]. [2003], 23.2.2011 [cit. 2012-04-24]. Dostupné z: http://fuse-emulator.sourceforge.net/libspectrum.php

[9] *PortAudio: Portable Cross-platform Audio I/O* [online]. [cit. 2012-04-29]. Dostupné z: http://www.portaudio.com/

[10] *The ZX Spectrum 48K Service Manual.* THORN (EMI) DATATECH LTD, 1984. Dostupné z: ftp://ftp.worldofspectrum.org/pub/sinclair/technical-docs/... ZXSpectrum48K_ServiceManual.pdf

[11] Visual 6502 in JavaScript. *www.Visual6502.org* [online]. [cit. 2012-03-25]. Dostupné z: http://www.visual6502.org/JSSim/index.html

[12] World of Spectrum: Archive. *World of Spectrum* [online]. [cit. 2012-03-16]. Dostupné z: http://www.worldofspectrum.org/archive.html

[13] World of Spectrum: Emulators. *World of Spectrum* [online]. [cit. 2012-04-27]. Dostupné z: http://www.worldofspectrum.org/emulators.html

[14] World of Spectrum: 16K / 48K ZX Spectrum Reference. *World of Spectrum* [online]. 18.9.2005 [cit. 2012-04-29].
Dostupné z: http://www.worldofspectrum.org/faq/reference/48kreference.htm

[15] wxHexEditor. *SourceForge.net* [online]. [2007], 1.3.2012 [cit. 2012-04-24].
Dostupné z: http://sourceforge.net/projects/wxhexeditor/

[16] wxHexEditor: SCM. *SourceForge.net* [online]. © 2012 [cit. 2012-04-24].
Dostupné z: http://sourceforge.net/scm/?type=svn&group_id=185385

[17] *WxWidgets: Cross-Platform GUI Library* [online]. [cit. 2012-04-29].
Dostupné z: http://www.wxwidgets.org/

[18] Z80ex. *SourceForge.net* [online]. © 1999-2009 [cit. 2012-04-24].
Dostupné z: http://z80ex.sourceforge.net/

[19] ZILOG WORLDWIDE HEADQUARTERS. *Z80 Family CPU User Manual*. San Jose (Kalifornie, USA), 2004.
Dostupné z: http://www.z80.info/zip/z80cpu_um.pdf

A Příloha na CD

K této práci je přiloženo CD se zdrojovými kódy aplikace a textem práce v elektronické podobě.

A.1 Obsah CD

- SUS107_DP2012_TEXT.PDF – text diplomové práce v elektronické podobě
- sus107−dt.tar.gz – archiv se zdrojovými kódy aplikace

A.2 Obsah archivu sus107-dt.tar.gz

- CMakeLists.txt – soubor pro sestavení aplikace nástrojem *CMake*
- **cmake_modules** – složka s pomocnými soubory pro nástroj *CMake*
- **lang** – složka s přeloženými soubory s lokalizací GUI
- **lib**/**z80ex** – složka se zdrojovými kódy knihovny *z80ex* (zdrojový kód třetí strany)
- **po** – složka se zdrojovými kódy pro lokalizaci GUI
- **src** – složka se zdrojovými kódy samotné aplikace (kromě níže uvedené komponenty *HexEditorCtrl* byl všechen kód v této složce a podsložkách vytvořen v rámci této práce)
- **src**/**ui** – složka se zdrojovými kódy GUI aplikace
- **src**/**ui**/**widgets**/**HexEditorCtrl** – složka se zdrojovými kódy komponenty GUI *HexEditorCtrl* (zdrojový kód třetí strany)
- **ui**/sus107−dt.wxg – soubor s návrhem GUI v nástroji *wxGlade*
- 48.rom – soubor s obsahem paměti ROM (z [3])

B Instalační příručka aplikace

Tato příručka popisuje kroky potřebné k sestavení aplikace ze zdrojových kódů. Předpokládá se, že aplikace instalována na počítači s operačním systém UNIX/Linux.

B.1 Požadované softwarové vybavení

Uvedená čísla verzí odpovídají těm, se kterými byla aplikace vyvíjena a testována. Aplikaci je možné sestavit i s jinými verzemi knihoven, avšak čím více se číslo verze liší, tím větší je pravděpodobnost výskytu nekompatibility.

- knihovna *wxWidgets* ve verzi 2.8.12 s podporou Unicode
- knihovna *libspectrum* ve verzi 1.0.0
- knihovna *PortAudio* ve verzi 19
- nástroj *CMake* ve verzi 2.8
- balík nástrojů *GNU GCC Toolkit* s kompilátory C/C++

B.2 Postup instalace

1. Rozbalení zdrojových kódů aplikace:

   ```
   tar xvf sus107-dt.tar.gz
   ```

2. Příprava souborů Makefile:

   ```
   cd sus107-dt
   cmake .
   ```

3. Zkompilování aplikace:

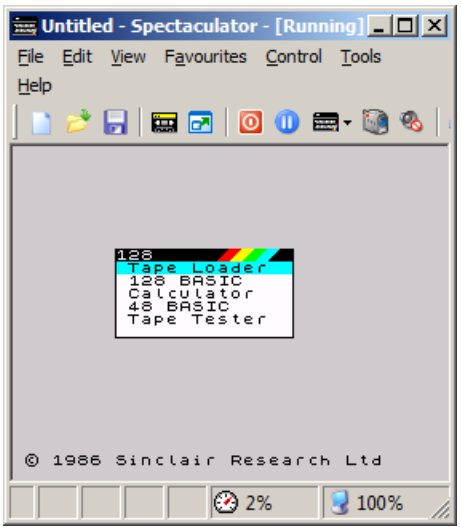
   ```
   make
   ```

4. Instalace (nepovinné):

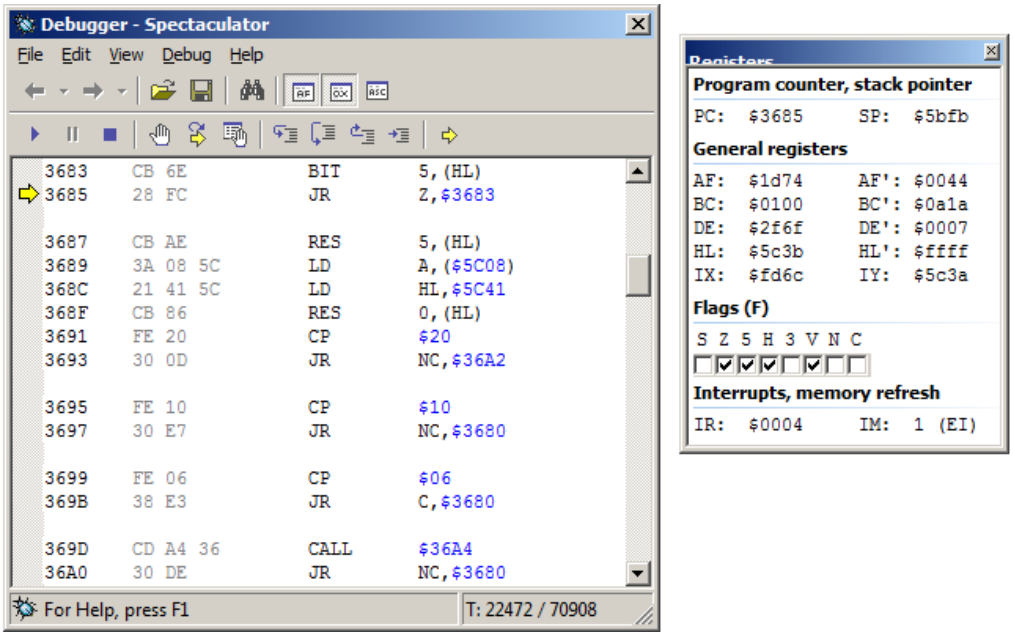
   ```
   make install
   ```

Po provedení kroku 3. je v aktuální složce sestaven spustitelný soubor `sus107-dt`. Aplikaci je možné spustit přímo z této složky bez nutnosti instalace do systému.

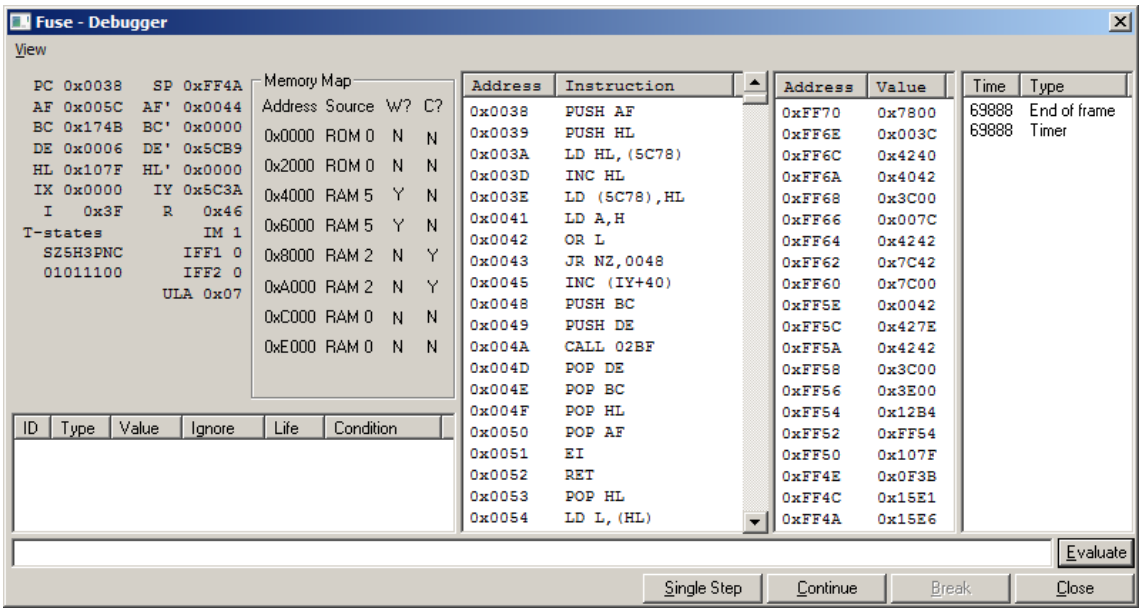
C Ukázky existujících emulátorů počítače ZX Spectrum



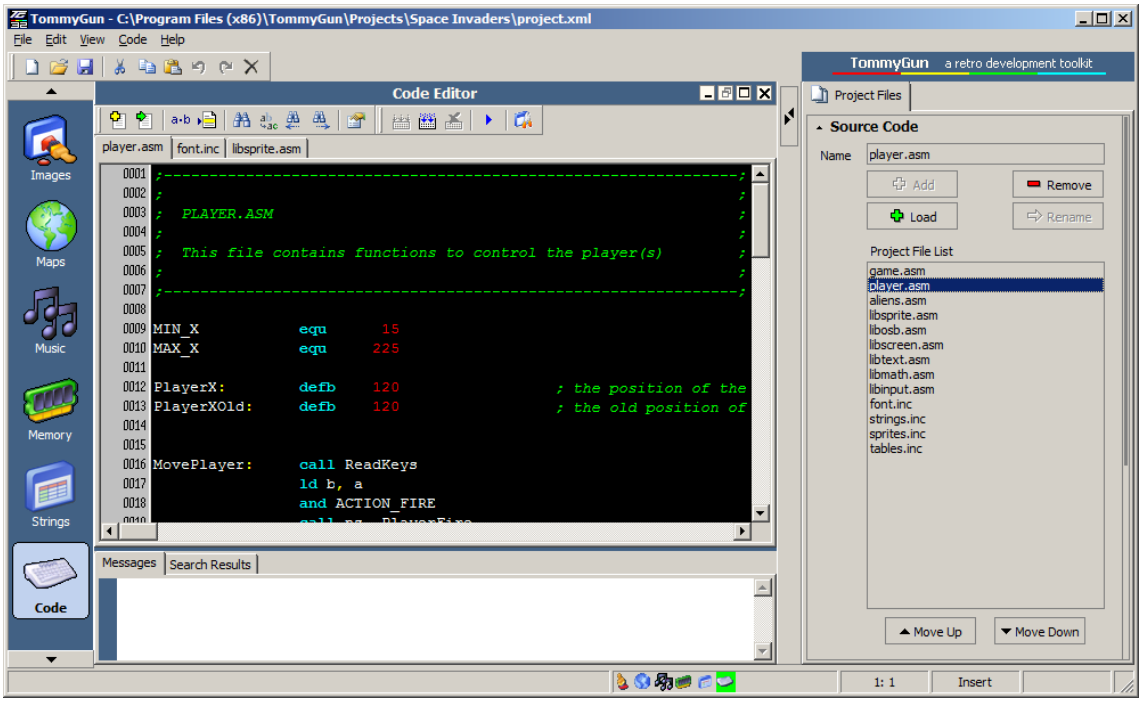
Obrázek 17: Základní obrazovka emulátoru Spectaculator



Obrázek 18: Debugger emulátoru Spectaculator



Obrázek 19: Debugger emulátoru FUSE



Obrázek 20: Vývojové prostředí TommyGun